SFA Modernization Partner

United States Department of Education

Student Financial Assistance

Integrated Technical Architecture

Detailed Design Document

Volume 2 – Internet Architecture

Task Order #16

Deliverable # 16.1.2

October 13, 2000

Table of Contents

List of Figures

List of Tables

1 Introduction

The Internet Architecture (IA) supports the development, execution, and operation of web browser-based applications. It is composed of several independent components that integrate together to provide an overall net-centric capability. The integrated solution provides a maintainable, extendible, manageable solution that encourages and facilitates reuse and reduces development time by leveraging the architecture. It provides value by adapting existing legacy applications and standardizes application development, allowing for significant flexibility and cost savings.

The IA is the part of the Execution Architecture that provides the user dialog through a Web interface. Users of the Integrated Technical Architecture (ITA) access the Student Financial Assistance (SFA) systems through personalized information portals. These portals provide context sensitive access to legacy applications and enable users to search content by relevant subject. The IA bridges both the Internet and Intranet user community to the legacy SFA environment using the Enterprise Application Integration (EAI) Architecture and Data Warehouse Architecture. The Security Architecture insures that the Internet Architecture restricts and permits access appropriately for the user's permission.

The following diagram illustrates the IA context within the ITA:

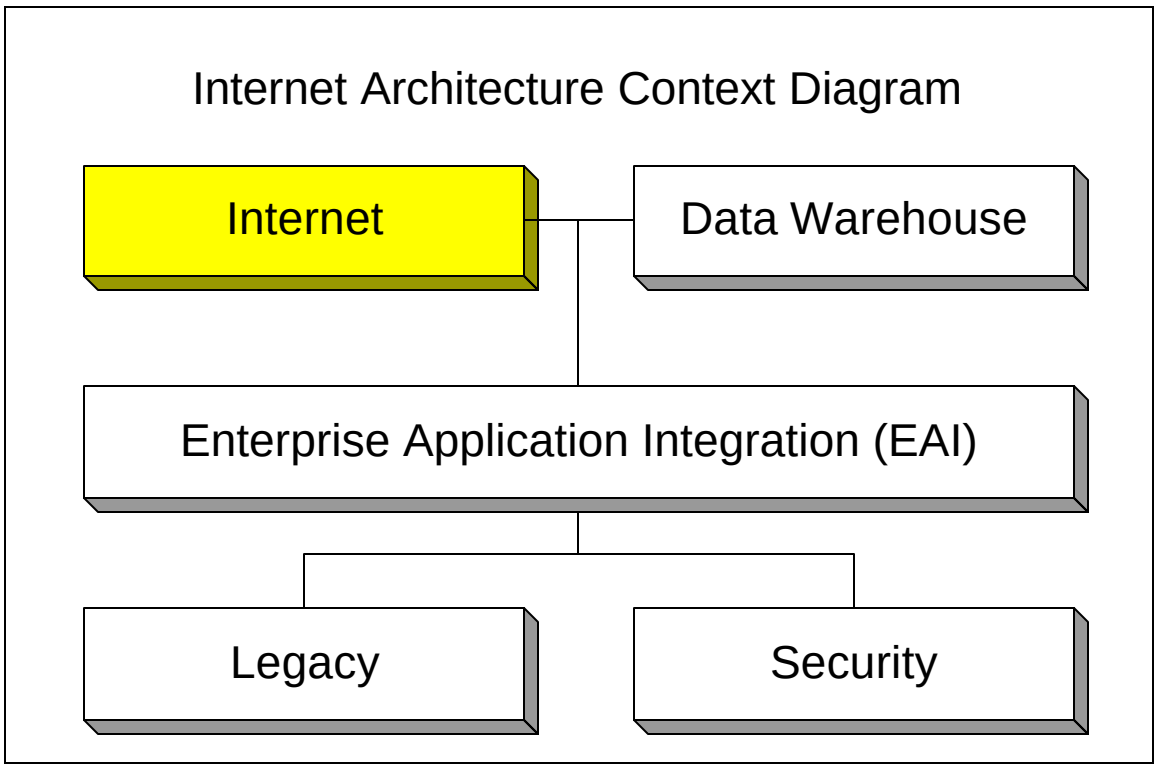

Figure 1 – Internet Architecture Context

2 Internet Architecture Overview

The IA is composed of equipment and commercial-off-the-shelf (COTS) applications. These components are specified and configured to provide superior performance and high availability.

The IA equipment consists of enterprise class servers. The servers utilized are Sun 3500 Solaris systems and other new technology (NT) based systems. The Sun 3500 Solaris systems are used for computationally intensive applications and applications requiring high availability. The NT based systems are used for applications that specifically require NT.

The IA applications consist of already existing Internet based applications augmented with other applications being deployed which interface with legacy enterprise SFA applications. The IA provides a way to integrate these applications through an information portal. Access to the already existing Internet based applications are through hypertext markup language (HTML) uniform resource locator (URL) links to the present Website. Other applications being deployed utilize the services of the IA as a function platform.

The IA COTS products consist of best-of-breed commercial applications that are integrated through standards-based application programming interfaces (API). Integrating these products through standards-based APIs enables other COTS products to be utilized when appropriate.

The IA framework is comprised of a set of components that provide the required services for a robust and secure Internet and Intranet environments. The principal functional components of the IA are listed in the following table.

Table 1 – Internet Arc hitecture Components

Component	Description
Web Browser	Allow users to view and interact with applications and documents made up of varying data types, such as text, graphics, and audio. Provides support for navigation within and across documents no matter where they are located, through the use of links embedded into the document content.
Firewall	Protects sensitive resources or information attached to a network from unauthorized access. A variety of firewall implementations may be required at various levels within the SFA network model.
Load Balancing	Distributes IP traffic across a set of SFA application servers to achieve high availability and predicable performance.
Web Server	Enable SFA to manage and publish information and deploy network-centric applications over the Internet (public) and Intranet (private) environments
Application Server	Extend SFA capability by supporting net-centric applications as well as providing an application architecture for enabling the development and execution of common services across different business capabilities.

Component	Description
Component Broker	An Object-Oriented enterprise solution for distributed computing, providing a scalable, manageable environment for developing and deploying component based solutions.
Content Management	Manages the Website content set in a process-oriented fashion using configuration control methods. It provides content versioning and supports both structured and unstructured content formats.
Portal	Provides a customizable and personalized view as a single access point to a wide variety of heterogeneous data sources.
Knowledge Management	Provides an informative searching and retrieval capability for both structured and unstructured content. Information that can be searched includes but is not limited to documents, spreadsheets, HTML-based files, e-mail messages and electronic news feeds.
Directory Server	Act as a central data repository that simplifies communication and sharing of resources. It allows diverse applications, machines, and users (both inside and outside the enterprise) to access consistent information and services. This simplifies such tasks as electronic-mail addressing, maintenance of computing environments, and user authentication and authorization.
File Storage	Implements a secure and persistent network file system. Caching is utilized to ensure efficient resource utilization. High availability is achieved through replication of file systems and encapsulation of Storage Area Network (SAN) resources.
Database Server	Responsible for providing access to the operational data store (ODS). Maintains integrity of the data within the database and supports the ability to store data on either a single physical platform, or across multiple platforms.

These components are discussed in detail in Section 5. The following diagram illustrates the principal components of the IA and lists the principal services they provide.

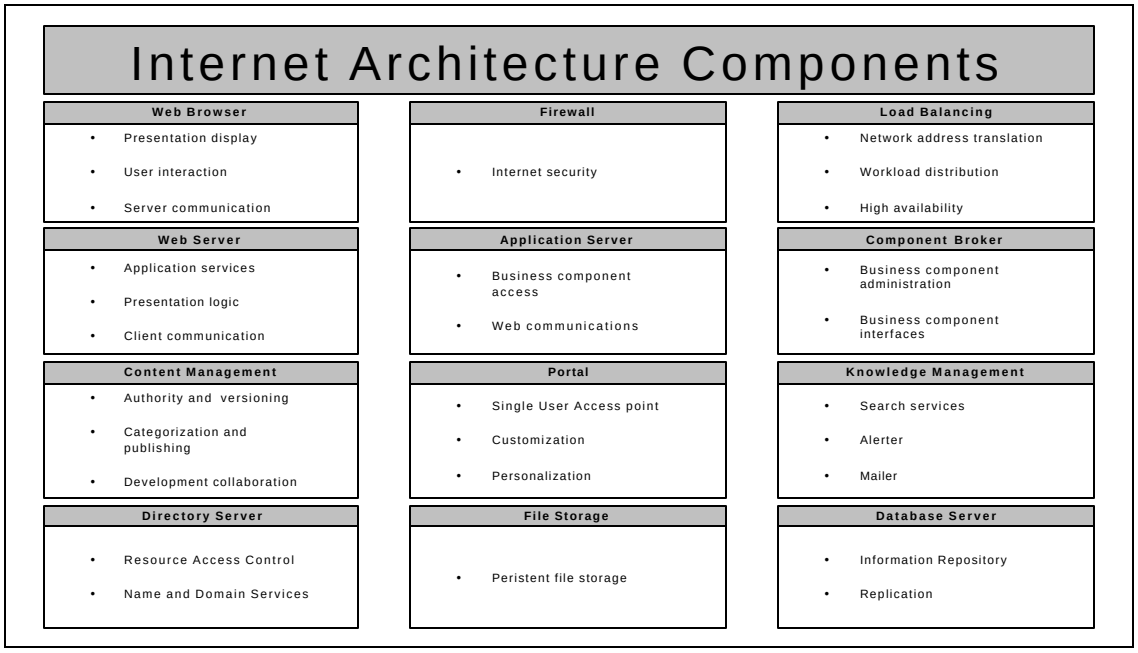

Figure 2 –Internet Architecture Components

3 Technical Overview

3.1. Requirements and Assumptions

The design of the IA is based on a critical set of technical requirements and assumptions. These requirements and assumptions determine the basic relationship of the individual IA components and the entire solution. The following table lists the principal requirements and assumptions, and presents their rational.

Table 2 – IA Requirements and Assumptions

Technical Requirements and Assumptions	Rationale
Scalability	The SFA solution supports both horizontal and vertical scaling. Horizontal scalability is achieved through application replication and load balancing. Vertical scalability is achieved through server chassis selection that allows both processing and data capacity expansion.
Reliability	The SFA solution achieves high availability through application and asset replication. Clustering techniques are not required for the SFA solution, although specific opportunities may require the use of clustering or clustering-like solutions.
Performance	The SFA equipment , COTS product and application configuration is based on engineering estimates derived from observation of existing systems and commercial best-practice implementation patterns typical of systems with equivalent user communities.
Security	The initial release of the SFA solution implements authentication via Viador and application specific means. Other applications and COTS products may utilize a Viador API to determine the authentication parameters. Subsequent releases of the SFA solution would implement a standards-based security architecture. Established commercial best-practice implements separate Internet and Intranet content via separate Websites. Although the information portal COTS product provides access control to content based on user assigned group, content isolation using separate assets and content partitioning is a superior defense against intentional and unintentional security penetration. Department of Education security policy prohibits any web-browser active (dynamic) components

3.2. Solution

The IA is implemented as a set of servers that are interconnected via the Virtual Data Center (VDC) local area network (LAN). The VDC environment supports the servers and provides for system management. Other VDC assets provide essential services such as storage and archive of storage.

The IA servers were selected and configured with redundant components that are essential for sustaining operation. These essential components are power supply, processor, storage, and input/output.

Reliable power is achieved by using redundant power supplies and supplying separate power to each power supply. The VDC is responsible for providing and maintaining separate sources of conditioned power to insure that a power interruption event is isolated to only half the power source and corresponding power supplies.

The use of symmetric processing insures that processor failure only degrades performance and does not disable the entire server. The selected server family supports modular processor components. The use of modular processor components within the same server family allows spare processors to be acquired and managed efficiently. The spare processors would be available to support recovery of any server. Recovery from a processor failure may require cycling the server but once cycled the server would be available.

Implementation of the system storage installs two types of storage – the first is storage available within the system and the second is storage available via network. The system installed storage units provide for the operating system and the storage required for operation and management of the system. The system storage units are redundant and the stored information is replicated in order to survive storage errors and general storage failure. Network storage is utilized for bulk storage. Network storage provides reliable information retention and supports sharing of information across servers.

Survivable input / output is achieved by using redundant adapters and configuring access to redundant networks. Using redundant adapters and having access to redundant networks also enables input / output balancing. Failure of an adapter or a network may impact performance but the application should persist. Recovery from an adapter failure is possible by switching network connections to an available adapter. Recovery from a network failure should be transparent with automatic recovery.

Internet and Intranet connectivity is provided by the VDC. The VDC maintains redundant Internet connectivity through separate service providers and arranges for adequate capacity.

The server equipment utilizes the VDC LAN for connectivity between the servers and other VDC equipment such as network storage and legacy systems. Connectivity between the servers for application specific traffic may require the deployment and configuration of dedicated sub-networks. The IA implementation is phased to support an initial application deployment with limited user community and a subsequent deployment to address other applications and users.

The following diagram illustrates the initial deployment of the IA equipment and the allocation of COTS products to the equipment.

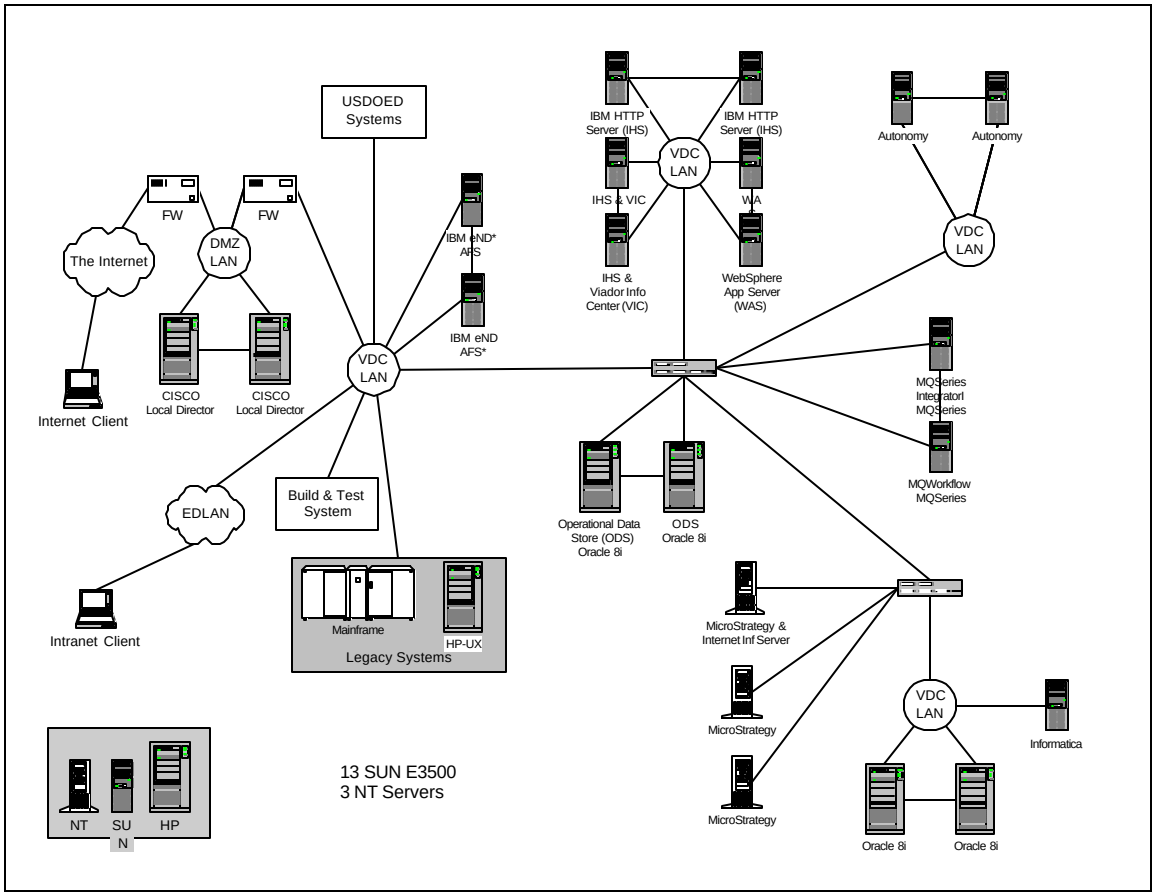

Figure 3 – IA Implementation

The following diagram illustrates a potential subsequent deployment of the IA equipment and the allocation of COTS products to the equipment. This configuration supports separate Internet and Intranet equipment and more extensive redundancy for critical servers. The separation of Internet and Intranet Website domains enhances security by physically separating the Website content and presenting a unique set of Website addresses.

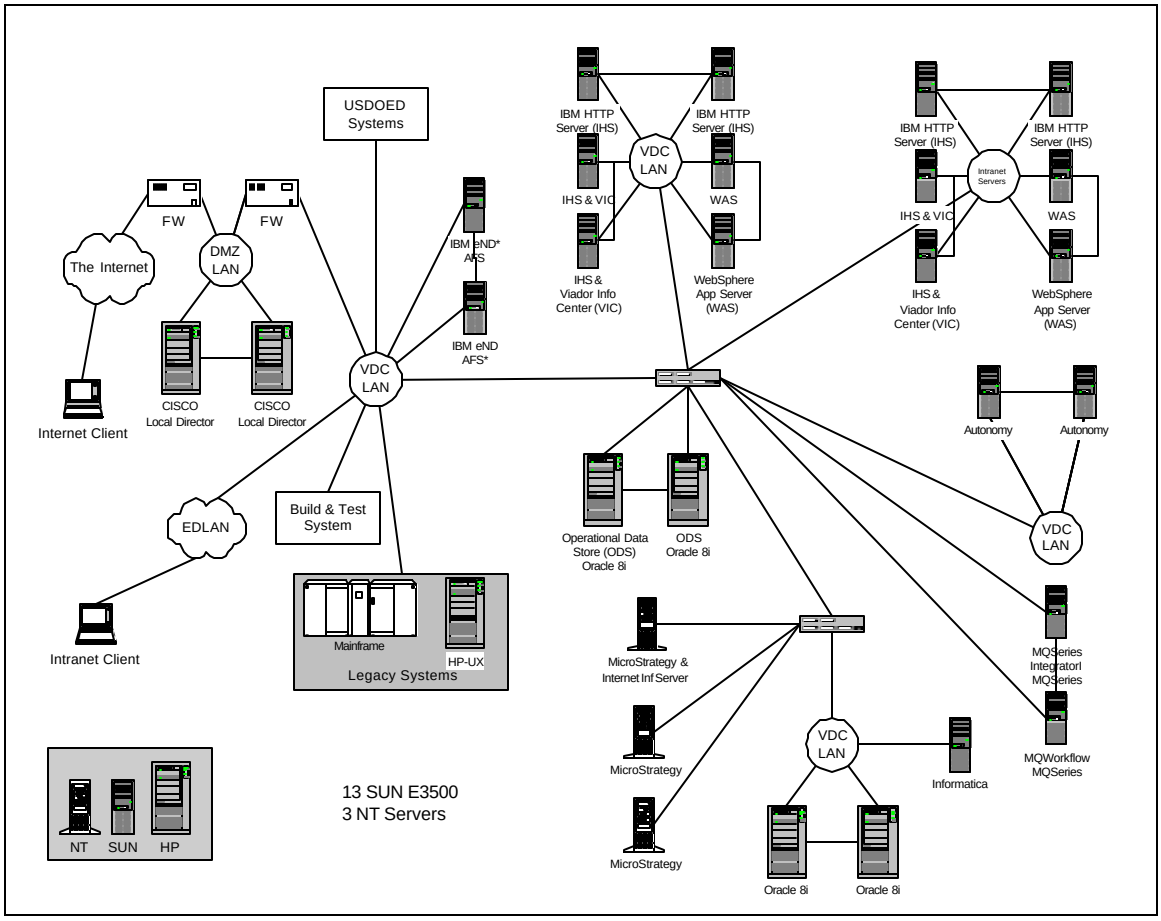

Figure 4 – IA Subsequent Implementation

The IA consists of 12 architectural components. The components partition the functional capabilities of the IA and provide a method to identify suitable COTS products to implement the IA. Some of the components are supplied by the Department of Education (DOE) or are native capabilities of the VDC. The following table lists each of the 12 architectural components of the IA, the COTS product or products selected to implement the component, and the respective source (IA or VDC) of the component. The specific version of each COTS product is specified in the table that follows.

Table 3 – IA Component COTS Products

Component		Product	Source
1	Web Browser	Microsoft Internet Explorer, Netscape Navigator, or Lynx	N / A
2	Firewall	Check Point FireWall-1	VDC
3	Load Balancing	IBM eNetwork Dispatcher	IA
4	Web Server	IBM HTTP Server	IA
5	Application Server	IBM WebSphere Application Server Enterprise Edition	IA
6	Component Broker	IBM Component Broker	IA

Component		Product	Source
7	Content Management	Interwoven TeamSite	IA
8	Portal	Viador Portal Suite	IA
9	Knowledge Management	Autonomy Knowledge Suite	IA
10	Directory Server	Netscape Directory Server	VDC
11	File Storage	IBM AFS	IA
12	Database Server	Oracle 8i	VDC

4 Technical Services and Invocation Methods

The IA components establish and provide functional services that enable the SFA applications. These services form a rich and efficient service-oriented layer and are the preferred methods for applications to invoke and utilize the IA. The following sections, organized by architectural component, identify and functionally describe the services. The principal architectural components are described and when appropriate the application invocation method is identified.

4.1. Web Browser

The Web Browser component is the user access mechanism to the IA. The component is a standard COTS product that implements the hypertext transfer protocol (HTTP) protocol, and renders HTML. The Web Browser services provide retention of the link connection, i.e., document physical location, and mask the complexities of that connection from the user. The following table identifies and describes the services provided by this component.

Table 4 – Services – Web Browser

Service	Functional Description
Server Communication	Utilizes standard protocols to establish a data connection to the server, transfer data, and terminate the connection.
Communication Security	Interoperable methods of securing the communications channel between the client and the server using secure protocols, e.g. SSL.
Presentation Services	Renders text, graphics, and other user interface components. Utilizes HTML as the content description language.

4.2. Firewall

The Firewall component protects SFA resources against direct and indirect intrusion. Access is managed using policy-driven restrictions on network connections, protocols, and data formats, and application-driven restrictions on data exchanges by applications and individuals.

Table 5 – Services – Firewall

Service	Functional Description
Packet Filtering	Protocol-based services check the address portion of data packets to determine the desired destination and intent. Administrators can block certain combinations that are categorized as unauthorized.
Proxy Services	Establishes a shielding or screening of the server address which is typically placed between the Internet router and the private network assets to be protected. Proxy servers shield users from knowing the specific addresses of servers within the private network.

4.3. Load Balancing

The Load Balancing component distributes workload across a set of applications and associated servers. The following table identifies and describes the services provided by this component.

Table 6 – Services – Load Balancing

Service	Functional Description
Network Address Translation	Distributes URL and other client-server workload across a set of servers using network address translation. Load balancing is achieved using various algorithms that account for resource availability in order to insure reasonable response time. Virtual address support allows the client request to be distributed across a set of servers and each client is only aware of the virtual address.

4.4. Web Server

The Web Server component manages document requests in formats such as HTML, PDF, etc.

The following table identifies and describes the services provided by this component.

Table 7 – Services – Web Server

Service	Functional Description
Client Communication	Uses HTTP to establish a data connection to the server, transfer data, and tear-down the connection.
Communication Security	Interoperable method of securing the communications channel between the client and the server using SSL.
Dynamic Page Services	Utilizes JSP to execute commands that are embedded in the presentation data to allow for run-time binding of presentation and data.
Application Services	A servlet based method to execute commands that interact with external systems and components, returning data to the requesting client.

4.5. Application Server

The Application Server component provides access to legacy systems, databases, and other application servers through a reusable and consistent application architecture. Applications are supported within a standards-based open development environment that provides the ability to use object-oriented technologies.

The following table identifies and describes the services provided by this component.

Table 8 – Services – Application Server

Service	Functional Description
Run-Time Services	A Java / CORBA compliant application environment. Java Virtual Machine (JVM) compliant with Java 1.2 or greater.

Service	Functional Description
Application Services	Programmatic specification of business logic in a reusable, component manner using an Enterprise JavaBeans implementation and behavior model.
Database Services	A JDBC API to database systems, regardless of database vendor.
Transport Services	Guaranteed message delivery service between components. A JMS / Javamail unified programming interface to external email and messaging servers, regardless of server vendor.
Directory Services	A JNDI facility used to discover remote network resources.
Transaction Management	A JTA based service allows multi-step processes to succeed or fail as an atomic unit.
Object Communications	Standard approach for objects to call the methods of other objects and RMI / IIOP ability to communicate across the network to CORBA objects.
Data Typing and Encryption	A JAF facility that helps components determine the data type of an arbitrary data stream, then encapsulate that stream into a known object format.

4.6. Component Broker

The Component Broker (CB) component provides implementation services to business objects or enterprise beans. Some of these object services are administrative in nature and their behavior is controlled by qualities-of-service configured through the management tools. Other services are presented to business object implementors as interfaces, and others are built into the infrastructure and work on behalf of the business logic.

For more information, see the International Business Machines (IBM) Component Broker *Advanced Programming Guide.*

The following table identifies and describes the services provided by this component.

Table 9 – Services – Component Broker Component

Service	Functional Description
Concurrency Control Service	The Concurrency Control Service consists of a set of interfaces that allow an application to coordinate access by multiple transactions or threads to a shared resource. When multiple transactions or threads try to access a single resource at the same time, any conflicting actions are reconciled so that the resource remains in a consistent state.
Event Service	The Event Service defines a channel between multiple objects which defines their roles and allows them to communicate asynchronously. There are two defined roles: supplier objects and consumer objects. Suppliers produce events, while consumers process events.
Notification Service	CB's Notification Service contains event channels that act as supplier and consumer objects. These event channels allow multiple suppliers to communicate with multiple consumers asynchronously and without confusing the many low-level details within the objects.

Service	Functional Description
Externalization Service	The Externalization Service provides a mechanism by which objects are able to save and restore their state in a non-object form. This allows the object's state to exist independently of the object itself.
Identity Service	CB derives an object identity from relative information that positions the object within its container, server, host, and domain. This information can be used within the CB Managed Object Framework to uniquely identify each object from any other object in the distributed system.
Life Cycle Service	A Life Cycle Service provides operations for creating, copying, moving, and deleting objects in a distributed environment. The Life Cycle Service in CB provides a level of abstraction between the client program creating an object and the determination of the location where that new object will exist.
Naming Service	The CB Naming Service allows you to create naming hierarchies so you can easily locate objects. In conjunction with other services, clients can navigate through different naming context trees to locate specific objects. CB Naming Service handles both absolute and relative paths.
Security Service	The Security Service is used primarily to prevent end users from accessing information and resources that they are not authorized to use. This predominantly covers distributed business objects, but by extension includes any of the information and resources from other non-object-oriented or non-distributed sources used by those business objects.
Transaction Service	The Transaction Service enables programmers to implement transactions by using standard object-oriented interfaces in a distributed environment. CB uses the Transaction Service to ensure that each application has correctly grouped the updates in the transaction so that the data is always updated consistently.
Session Service	The Session Service provides detailed information for applications in a distributed object environment to control the extent of a session and the application profile and arbitrary session properties that are relevant within the scope of that session. The scope of the session is defined to exist between the point when the session is started and the point when the session is ended.
Query Service	The Query Service enables you to find objects in a CB collection based on a set of conditions described with an object-oriented structure query language (OOSQL). The OOSQL enables you to describe complex search criteria. It is a extension of SQL with features for handling object collections, object attributes, and methods in query statements.
Cache Service	The Cache Service enhances concurrency and performance by supporting optimistic and pessimistic caching of data. In optimistic caching, frequently used data is cached in the memory of the CB server and not reread from the database on each transaction. Cached data is invalidated based on a time-out value.
Workload Management	The Workload Management capability allows the CB run time to dynamically allocate an application server to process a request. As more clients use an application, the amount of work increases and the load on the servers increases. The key to workload distribution in CB is the use of a server group to define multiple application servers with a common configuration.

4.7. Content Management

The Content Management component manages Website content delivery from the development environment to the production environment.

The following table identifies and describes the services provided by this component.

Table 10 – Services – Content Management

Service	Functional Description
Authoring	Allows users to associate and launch development applications against the content managed by the component.
Versioning	Maintains versions of each individual Website content artifact. The individual content versions are associated with Website configurations or releases.
Categorization & Publishing	Manages groups of content artifacts according to user defined criteria and supports publishing of these content artifacts.
Development Collaboration & Workflow	Provides process control and related methods that support collaboration between personnel in the development community and production community. The collaboration and workflow utilities provide a methodical way to insure that content change is appropriately authorized.
Integrates Multiple File Types	Any file type is supported. The Interwoven product is not aware of or dependent on the file type.
Summarization	Produces a summary report of a configuration or release and the Website content artifacts that were delivered from the development environment to the production environment.

The following table identifies the interfaces between the Interwoven COTS product that implements the Content Management component and the other IA components.

Table 11 – Interfaces - Content Management Services Interfaces

Service	Invoked	API	Results of call
Deploy content	Open Deploy	Perl Script	Delta file created
DAS	Automatically	N / A	Updated content

4.8. OpenDeploy

Interwoven OpenDeploy is an enterprise-class content replication solution for the Website content. It provides a secure, flexible and scalable solution for cross-platform, transactional content transfer from development servers to production servers.

OpenDeploy allows a secure reliable method to transfer files to Development and Production servers. It has a “rollback” feature to quickly regenerate previous version of a Website, and can be utilized to schedule, automate and synchronize content deployment activities.

4.8.1. DataDeploy

The interface between TeamSite and the database server works with Templating to update Database server from metadata captured in Template, and utilizes a DBD module to communicate with Oracle.

4.8.2. Autonomy Interface

It is possible to interface Autonomy with OpenDeploy by using an external task. This script can launch the Autonomy application to spider content that had recently been deployed.

Another approach is scheduled deployments. OpenDeploy can be scheduled to put content onto the Web at certain intervals. The Autonomy product can spider the content after all the content has been updated. Since the content is going to be relatively small updates should not take large amounts of time or resources.

4.8.3. Viador Interface

There is no need for an interface between these products.

4.9. Portal

The Portal component provides a customizable and personalized interface as a single access point to a wide variety of heterogeneous data sources including Website content, documents, and existing applications.

The following table identifies and describes the services provided by this component.

Table 12 – Services – Portal

Service	Functional Description
Single Point Of Access	The portal provides a single interface in which to access a wide variety of heterogeneous data sources within SFA. These heterogeneous data sources can consist of the following: structured data such as content documentation, unstructured data such as intranet and Internet Web content and existing enterprise applications.
Customization	The portal can be customized to provide access to a range of data sources and applications based on the users and their roles. These roles can be defined by the SFA system administrator and then applied to the categories of SFA users such as students, schools, etc.
Personalization	The portal can be personalized by selecting from the catalog of data sources and applications accessible according to the assigned roles of the SFA users. This allows the SFA users the ability to quickly locate required information and filter extraneous information.
Authorization	The portal provides usage authorization to control the level of granular access an SFA user has to the portal itself. This is organized by individuals, roles, or groups as defined by the SFA system administrator.
Authentication	The portal provides authentication as the process of uniquely identifying a specific SFA user and maintaining the accessibility of information as defined in the customization performed by the SFA system administrator.

The following table identifies the APIs for the services provided by this component.

Table 13 – APIs – Portal Services

Service	API	Functionality
Customization	Viador Repository (VR)	This Java API provides access to Viador Repository information in order to support customization services such as creation and maintenance of users, groups, channels, etc.
Customization	Viador Portal Customization (VPC)	This JavaScript API provides access to Viador Repository information in order to support customization services such as creation and maintenance of users, groups, channels, etc.
Configuration	Viador Portlet (VP)	This API provides for the integration of third-party applications by providing a skeleton for creating an application as a portlet.
Authentication	Viador Open Authentication (VOA)	This API provides for the integration of the portal with other authentication systems.
Searching	Viador Data Feed (VDF)	This Java API provides access to the search engines and document management systems for the portal.

4.10. Knowledge Management

The Knowledge Management component provides the information search and retrieval capability. This component offers various search types on different data groups as unstructured digital information, structured data, word processing documents, HTML-based files, e-mail messages and electronic news feeds. This product is able to support a thesaurus query if a thesaurus is loaded into the environment.

The following table identifies and describes the services provided by this component.

Table 14 – Services – Knowledge Management

Service	Functional Description
National Language Search	A search capability specified as a subject / verb criteria.
Boolean Search	A structured search capability specified as a Boolean or logical expression criteria.
Proximity Search	A search capability based upon a text delta algorithm.
Proper Names Search	A search capability restricted to proper names.
Simple keyword	A search capability based on a key text string.
Query Search	A search capability based on user queries.

Service	Functional Description
Bracketed Boolean Search	A search capability specified as a continuation of Boolean or logical expression contents.
Mailer	A scheduled process that queries the DRE and sends the results to the users either as a list of links to content in a single message, or the content itself in a separate messages.
Alerter	A feature that will direct a large flow if information to those individuals who have an interest in it.

4.11. Directory Server

The Directory Server component manages information common to applications, individuals, and groups of individuals.

The following table identifies and describes the services provided by this component.

Table 15 – Services – Directory Server

Service	Functional Description
Name Services	A logical component of directory services provided to create a logical "pronounceable" name in place of a binary machine number. These services are used by other communication services such as file transfer, message services, and terminal services.
Domain Services	Provide a mechanism by which various nodes are recognized. These services use the domain portion of an address to transport the data to the corresponding node. Therefore, Domain Services are functions that track and recognize different logical organizations and then map them to physical resources as tracked by the Naming Services.
Single Sign-on Services	Supports a single sign-on capability by providing a common user authentication repository and a standards-based access method. Single sign-on is actually implemented in association with other products, but the essential framework is LDAP.
Personalization Preferences	Individual and groups of individuals may be associated with preferences. This capability manages these preferences on both a global basis and per application
Authentication	User sign-on is verified using a password. User passwords are administered with security controls.
Access Control	User access to applications and specific files may be controlled. Access controls may be allocated and enforced for individuals or for a group of individuals.

4.12. File Storage

The File Storage component provides an IA file system hierarchy that utilizes the VDC SAN. The Andrew File System (AFS) is used to achieve performance and high availability.

The following table identifies and describes the services provided by this component.

Table 16 – Services – File Storage

Service	Functional Description
Fileserver	Handle requests at the file and directory level.
Volume Management	Handles operations at the volume level.
Consistency Checking	Checks the system for internal consistency and repairs errors it finds.
Authentication	Responsible for maintaining the Authentication Database.
Access Control	Responsible for maintaining the Protection Database.
Cache Management	Responsible for maintaining the database and for providing the Cache Manager with information about volumes and volume location.
File System Backup	Responsible for maintaining the Backup Database and for providing an interface to the AFS Backup System.
File System Synchronization	Responsible for transferring information from System Control Machines (SCM) and Binary Distribution Machines (BDM) to other AFS servers.
Directory Distribution	Responsible for distributing the contents of a specified directory.
Time Synchronization	Synchronizes an AFS server's clock with the clock on another machine.

4.13. Database Server

The Database Server component provides a consistent relational interface to information contained in a database. The component supports high performance storage and retrieval of structured data.

The following table identifies and describes the services provided by this component.

Table 17 – Services – Database Server

Service	Functional Description
Storage Services	Manage data physical storage. These services provide a mechanism for saving information so that data will live beyond program execution. Data is often stored in relational format (an RDBMS) but may also be stored in an object-oriented format (OODBMS) or other formats such as IMS, VSAM, etc.
Indexing Services	Provide a mechanism for speeding up data retrieval. In relational databases one or more fields can be used to construct the index. So when a user searches for a specific record, rather than scanning the whole table sequentially the index is used to find the location of that record faster.
Security Services	Enforce control regarding which records authorized users can view and edit, and which functions they can execute. Most database management systems provide data access control at the database, table and row levels, and execution control for stored procedures, database functions, etc. to specific users and groups.
Access Services	Enable an application to retrieve data from a database as well as manipulate (insert, update, delete) data in a database. SQL is the primary approach for accessing records in today's database management systems.
Replication Services	Support an environment in which multiple copies of databases must be maintained.

5 Detailed Design Specifications

The SFA ITA consists of an integrated set of COTS products. This integration is described by elaborating each COTS product interface, or touch-point, with other COTS products and with the SFA applications. The following material describes the architecture as a partition of service clusters and further describes the interface between these clusters via scenarios. These scenarios model the information flow through the touch-point.

5.1. ITA Service Cluster Architecture

The ITA architecture provides a high availability solution for the implementation of SFA applications. This architecture is partitioned across a set of Web Server / Application Server (WS / AS) clusters to specifically support IBM HTTP Server / WebSphere Application Server (IHS / WAS) and COTS product integration in a consistent fashion and to achieve efficient resource load balancing. This architecture achieves high availability using the IBM SecureWay Network Dispatcher (ND) COTS product to manage Web-site traffic. ND utilizes performance metrics to load balance SFA application Web-site traffic and automatically senses WS / AS cluster resource failure to route application traffic to the available cluster resource.

Other clusters are used to achieve high availability for critical application services. The Release 1 architecture and design utilizes a ND cluster for the Web-site search engine service. The search engine cluster design operates in a similar manner as the WS / AS cluster, where the SFA application behaves as a Web Browser component and the search engine behaves as a Web Server component.

The Release 1 architecture and design consists of the following ND clusters listed in the following table.

Table 18 – ITA Service Clusters

ND Cluster	Description
Portal Cluster	The portal cluster consists of a redundant IHS and Allaire JRun COTS product application server (AS) configuration. The JRun AS is utilized because of specific product integration requirements of the Viador portal COTS product. This cluster configuration is unique to Release 1 and is not required once Viador is migrated to the WAS environment.
Application Cluster	The application cluster is a redundant IHS and WebSphere Application Server (WAS) configuration. This cluster is the standard WS / AS solution for SFA applications. Both the IFAP and Intranet R2.0 SFA applications utilize the application cluster.
Content Management Cluster	The content management cluster consists of a non-redundant Apache WS integration with the Interwoven TeamSite COTS product. The Apache WS is utilized for this cluster because of specific product compatibility issues. Upon resolution of the compatibility issues TeamSite would be integrated with IHS. A redundant configuration is not required because this application service is not critical. Recovery is achieved using alternative resources.

ND Cluster	Description
OLAP Cluster	The OLAP cluster consists of a redundant Microsoft IIS and MicroStrategy COTS product configuration. The non-standard Microsoft IIS solution is specifically required for the MicroStrategy Active Server Page implementation.
Autonomy Cluster	The Autonomy cluster consists of a replicated DRE configuration. This cluster is an application service that has a HTTP interface. This interface enables the ND to load balance the Autonomy resources.

The following diagram illustrates the ND clusters.

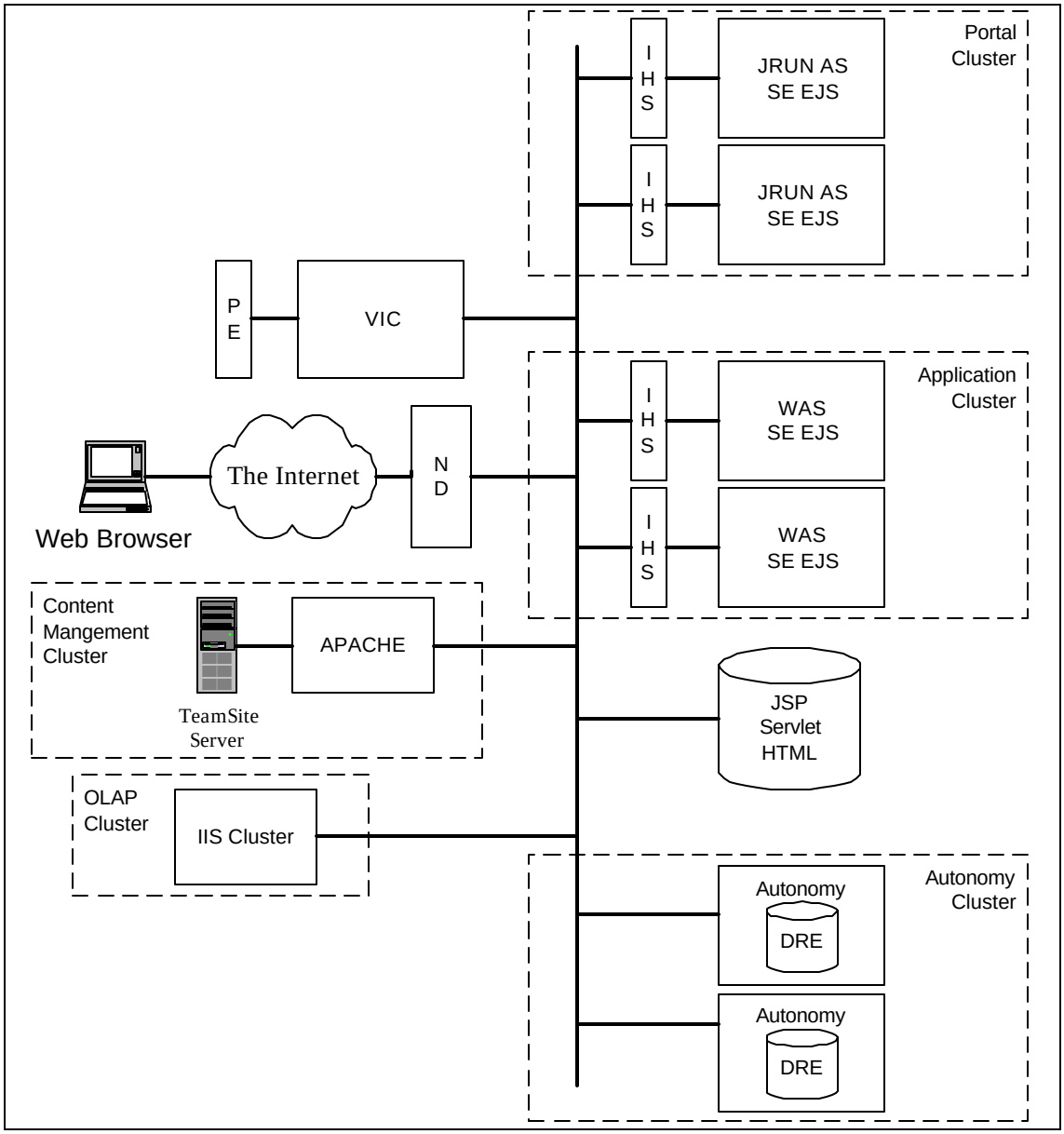

Figure 5 – ND Cluster Architecture

5.2. ITA Touch-Point Scenarios

The following touch-point interfaces are used to show how the service clusters operate to support the SFA applications.

- Touch-Point # 1 (Portal – SFA Application)
- Touch-Point # 2 (SFA application – Autonomy)
- Touch-Point # 3 (IHS – WAS static HTML)
- Touch-Point # 4 (IHS – WAS Servlet Flow)
- Touch-Point # 5 (IHS – WAS JSP Flow)
- Touch-Point # 6 (IHS – WAS EJB Interaction)
- Touch-Point # 9 (Autonomy – Document Content Publishing)
- Touch-Point # 10 (Autonomy – URL Content Indexing)
- Touch-Point # 11 (Autonomy – High Availability Configuration)

5.2.1. Touch-Point # 1 (Portal – SFA Application)

The Touch-Point #1 interface design describes the basic behavior of the service cluster architecture using a scenario that traverses the principal service clusters. The scenario traces information flow through the Schools Portal, SFA applications, and the search engine service. This interface is for SFA applications that are IHS / WAS based or are separate application programs.

The following diagram illustrates the scenario.

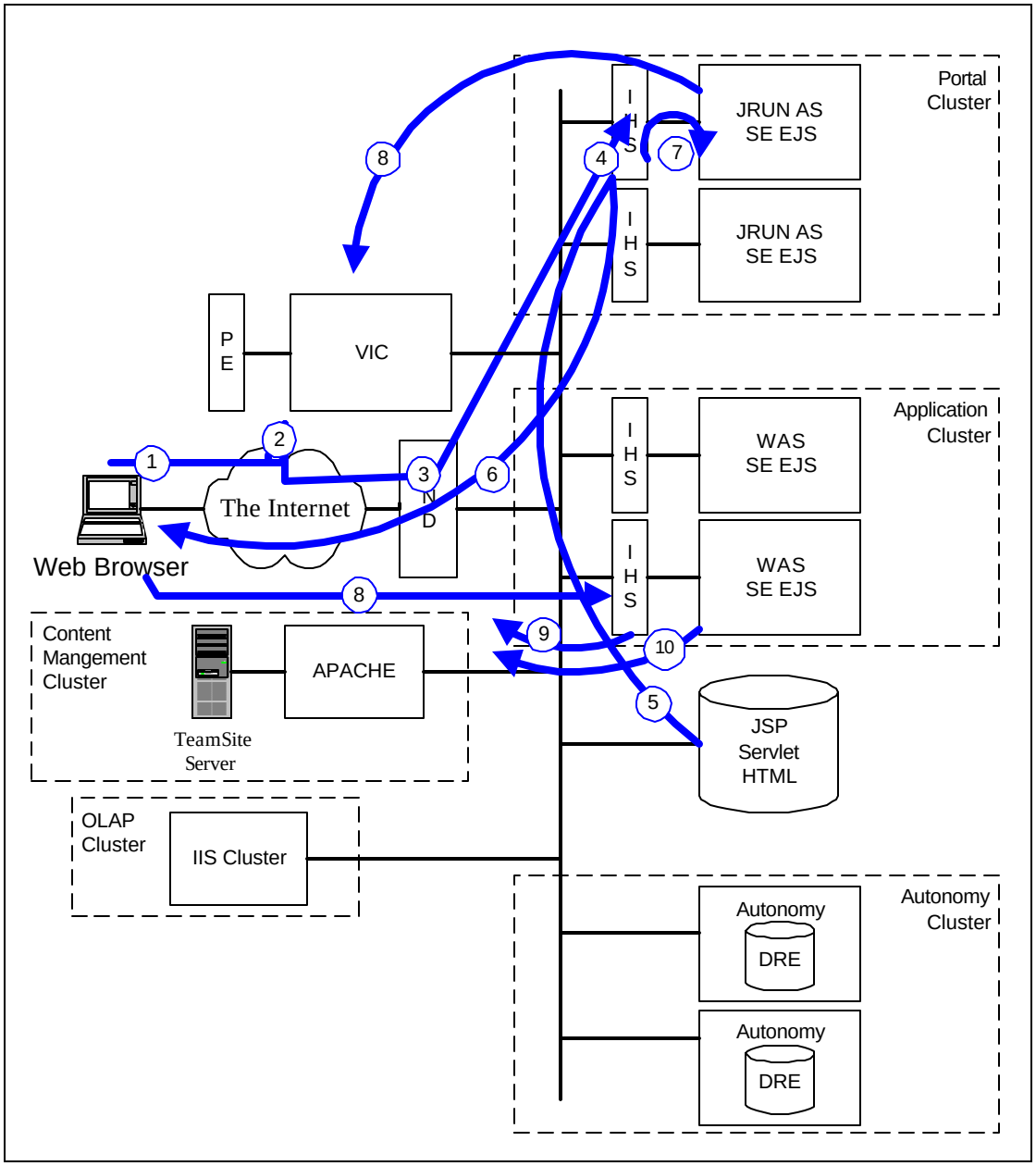

Figure 6 – Touch Point # 1 – Detail Diagram: Portal SFA Application

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 19 – Touch Point # 1 – Step-by-Step Description

Step	Description
Step 1	A Schools Portal user specifies the HTTP URL of the SFA Schools Portal Main Page (e.g. www.schoolsportal.sfa.ed.gov\index.html).
Step 2	The users Schools Portal HTTP URL is resolved by the ed.gov DNS to the ND portal cluster virtual IP address.
Step 3	The Schools Portal HTTP request is routed by ND to one of the two IHS-WAS portal cluster servers. This IP route decision is based upon the availability and performance of the IHS-WAS servers.
Step 4	The Schools Portal HTTP request is received by the appropriate IHS and either specifies static HTML, a JSP, or a servlet.
Steps 5	For static HTML, IHS immediately finds the HTML in the HTML directory.
Step 6	IHS returns the HTML to the requesting user for Web Browser rendering.
Step 7	For a JSP / servlet, IHS forwards the JSP / servlet request to the JRun Application Server (AS). With subsequent JSP / servlet processing, the resulting HTML is returned to the requesting user for Web Browser rendering.
Step 8	The JSP / servlet request may utilize other application services such as the Viador Portal and / or the Autonomy Search Engine. Other SFA Applications may be accessed via the user Web Browser by specifying the SFA application HTTP URL, or indirectly by selecting an HTTP URL within an HTML page. Either way, the HTTP URL is resolved to the application cluster.
Steps 9 & 10	The application cluster either responds with static HTML or with dynamic HTML produced via JSP / servlet processing. The SFA application may invoke application services using JSP / servlet processing.

5.2.2. Touch-Point # 2 (SFA application – Autonomy)

The Touch-Point #2 interface design elaborates the method utilized by SFA applications to invoke search engine services. This interface is for SFA applications that are HIS / WAS based or are separate programs.

In Release 1, an SFA application invokes the search engine service using the Autonomy Knowledge Builder (KB) API. This API provides a C language interface, a Java language interface, and an HTTP interface. Both the C and Java API produce HTTP equivalent to the HTTP API.

In Release 2, the search engine KB API may be encapsulated as a JavaBean or a business object. Encapsulation of the KB API would simplify the invocation and provide a more functional SFA business capability.

The following example servlet fragment shows the Java invocation of the KB API for a trivial search engine request. The fragment performs a boolean search of an application DRE. The Autonomy cluster is specified via a domain name (e.g. autonomy.production.sfa.ed.gov).

```
package itso.servjsp.servletapi;
import com.autonomy;
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HTMLFormHandler extends HttpServlet {

public void init(ServletConfig srvCfg) throws ServletException {
         super.init(srvCfg);

public void doPost(HttpServletRequest req, HttpServletResponse res)
         throws ServletException, IOException {

    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML><TITLE>Site Search</TITLE></BODY>");
    out.println("<H2>Site Search</H2><HR>");

    out.println("<HR>");
    getSearch(req, out);
    out.println("</BODY></HTML>");
    out.close();
}

public void getSearch (HttpServletRequest req, PrintWriter out)
         throws ServletException, IOException {

    dreQuery myQuery;
    t_dre myDre;
    StringBuffer sBuffer;
    DreQueryResult myResults;

    String sAutonomy;
    String sAutonomy_IP;
    String aSearch_Argument;
```

```
    // Establish access to the Dre
    myQuery = new dreQuery ();

    // Configuration identifies 'Domain Name' of the Autonomy cluster
    sAutonomy := getAutonomy(); // e.g. "autonomy.production.sfa.ed.gov"

    // Resolve Autonomy cluster 'Domain Name' to IP using ed.gov DNS
    sAutonomy_IP := getAutonomy_IP(sAutonomy); // e.g. "204.98.12.7"

    // Obtain the Autonomy search argument
    sSearch_Argument := req.getParameter("searchargument");

    myDre := myQuery.dreCreateDre (sAutonomy, sAutonomy_IP, 7000, 7001);

    try
    {
        // Query the Dre
        myResults = myQuery.dreDoQueryToObject (
            myDre,
            myQuery.DRE_TEXT_QUERY,
            sSearch_Argument,
            "",
            "",
            "",
            ""
            10,
            myQuery.DRE_FULL_RESULTS,
            50,
            false);
        out.println("<H4><B>Search Results:</B></H4>");
        out.println ("(" + myResults.nResults + ")");
    }
    catch (DREException E)
    {
        out.println ("DRE Error occurred while trying to query DRE: "
            + toString ());
    }
}
}
```

Figure 7 – KB API SERVLET FRAGMENT

The following diagram illustrates the processing steps involved when an SFA application invokes the search engine service. An IHS / WAS application is assumed.

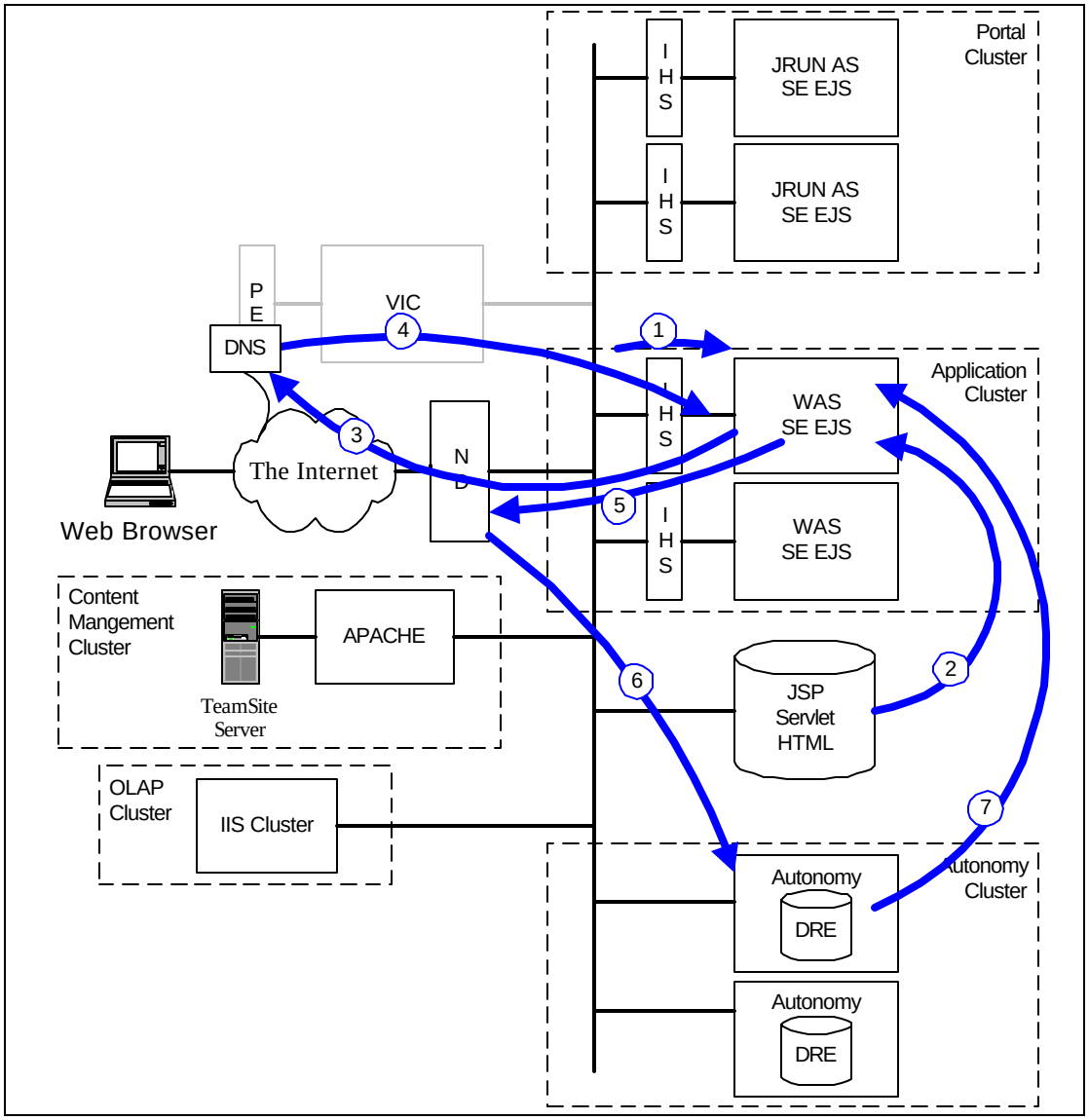

Figure 8 – Touch Point # 2 – Detail Diagram

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 20 – Touch Point # 2 – Step by Step Description

Step	Description
Step 1	An SFA application is invoked via an HTTP request, either from the SFA Portal or from an HTTP URL specified via a Web Browser.
Step 2	IHS / WAS interprets the HTTP request and invokes an SFA application JSP / servlet. The invoked application JSP / servlet performs application specific processing and subsequently invokes the search engine service either directly via the KB API, or indirectly using a JavaBean or Business Object.

Step	Description
Step 3	The KB API produces an HTTP request. The search engine service HTTP request URL domain name specifies the Autonomy cluster (e.g. autonomy.production.sfa.ed.gov). The KB API resolves the domain name using the DNS,
Step 4	The Autonomy cluster IP address is returned.
Step 5	The HTTP request is issued to the ND Autonomy cluster virtual IP address.
Step6	ND routes the HTTP request to an available Autonomy DRE IP address based on load balancing criteria.
Step 7	The Autonomy DRE accepts the HTTP request and performs the search. The search result is returned to the requesting JSP / servlet.

5.2.3. Touch-Point # 3 (IHS – WAS static HTML)

The Web-based SFA applications produce HTML that is rendered by a client Web Browser. The HTML may be static (pre-generated as part of the SFA application) or may be dynamic (generated in real-time through SFA application processing). This touch-point scenario describes the sequence of events for the client Web Browser and IHS – WAS interface with static HTML.

A client Web Browser initiates an SFA application request by URL. The Web Browser processes the URL, identifying a Web Server and a specific HTML artifact, and issues an HTTP request. The HTTP request is processed by the Web Server which gets and returns the specific HTML artifact to the requesting Web Browser. ND (not illustrated) manages the Web Browser to Web Server HTTP traffic to provie fault-tolerance.

The following diagram illustrates the basic information flow for this scenario.

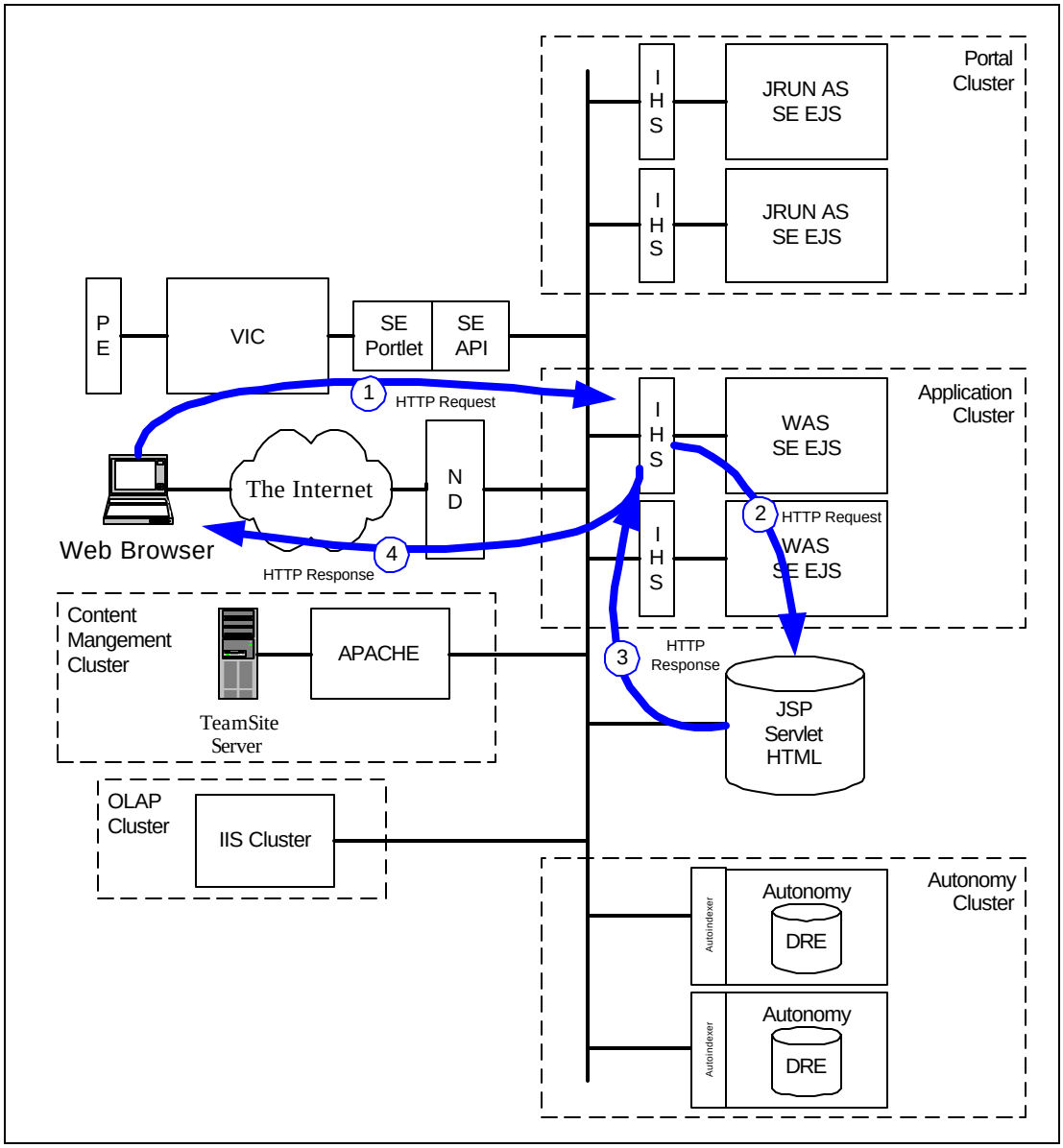

Figure 9 – Touch Point # 3 – Detail Diagram: Static HTML Flows Within WebSphere Application Server (WAS)

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 21 – Touch Point # 3 – Step by Step Description

Step	Description
Step 1	The client requests information using a Web Browser by specifying a URL. An HTTP Request is sent to the Web Server.
Step 2	The Web Server passes the HTTP Request for static content.
Step 3	Static content is sent as the HTTP Response (e.g. HTML) back to the Web Server.

Step	Description
Step 4	The Web Server passes the HTTP Response (e.g. HTML) back to the client Web Browser.

5.2.4. Touch-Point # 4 (IHS – WAS Servlet Flow)

In this scenario, when the Web Server receives a request for a servlet it redirects the request to the servlet repository. WebSphere then loads the correct servlet from the servlet repository (actually a directory on the application server classpath, typically <ServletRoot>\servlet) into the servlet engine and runs it. The input to the sevlet will be the HTTP request originally sent to the Web server by the client. The servlet output is usually an HTML output stream. They may also make calls to Java Beans, Enterprise Java Beans or even other servlets to obtain data.

The following diagram illustrates the information flow for this scenario.

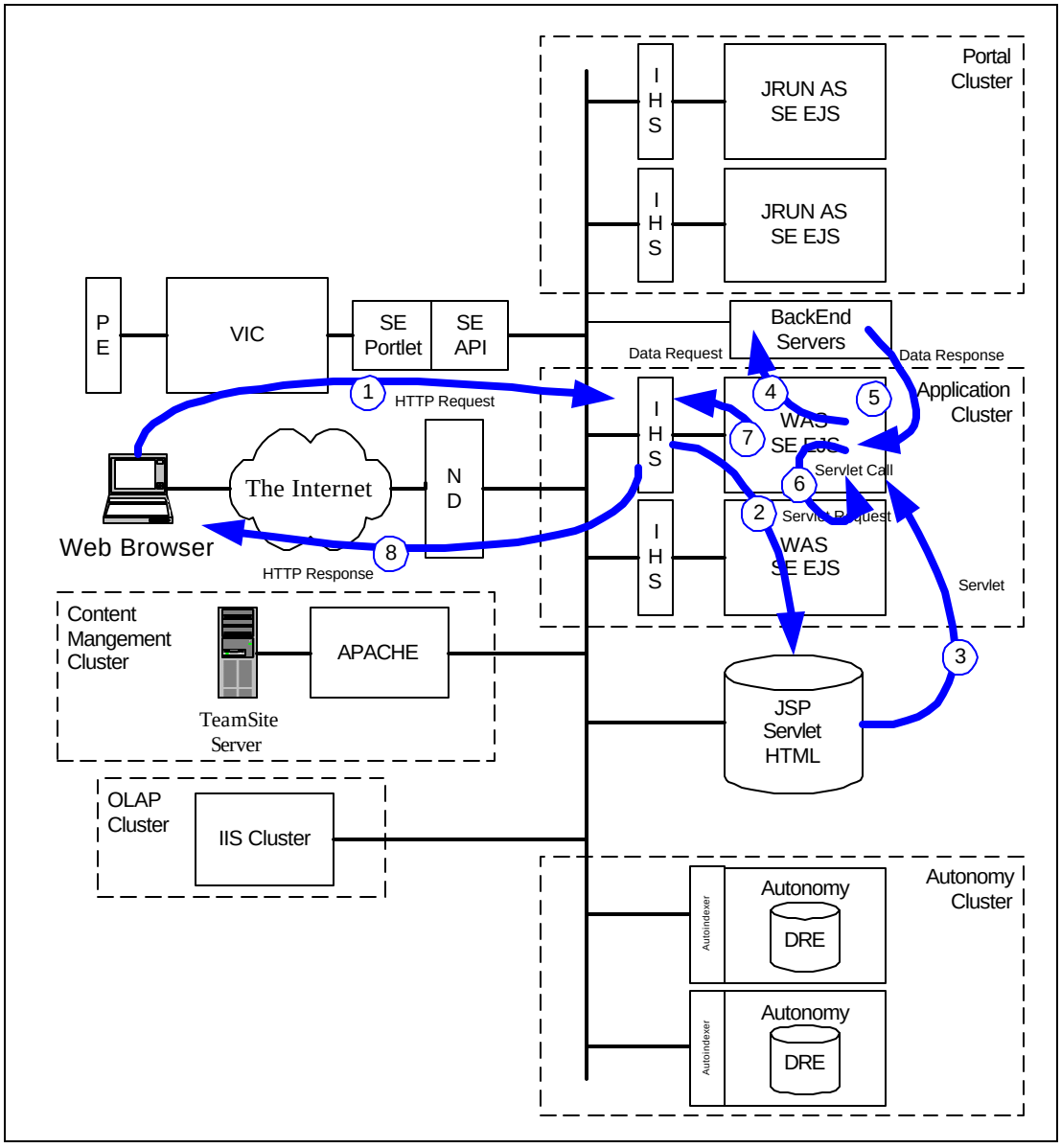

Figure 10 – Touch Point # 4 – Detail Diagram: Servlet Flows Within the WebSphere Application Server (WAS)

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 22 – Touch Point # 4 – Step by Step Description

Step	Description
Step 1	The client Web Browser sends an HTTP Request to a Web Server.
Step 2	The Web Server sends a servlet request to the servlet repository.
Step 3	The servlet engine processes the servlet.
Step 4	The servlet engine requests data from a backend server.
Step 5	The backend server sends the requested data.
Step 6	The servlet engine continues to process the servlet.
Step 7	The servlet produces HTML that is returned to the Web Server.
Step 8	The Web Server returns the HTML to the client Web Browser.

5.2.5. Touch-Point # 5 (IHS – WAS JSP Flow)

In this scenario, the HTTP request is processed though the web server (the diagram is a logical entity only). JSP source files are stored in the web server document hierarchy just like static HTML files. If the JSP consists only of HTML tags, the servlet produced by the compiler simply sets the correct fields on the response, opens an output stream to the client and writes the HTML. If other JSP tags are used then the compiler will create Java code in the servlet to perform the requested functions as well as writing the static parts of the HTML. The other tags may include java code fragments or directives to access back-end servers.

The following diagram illustrates the information flow for this scenario.

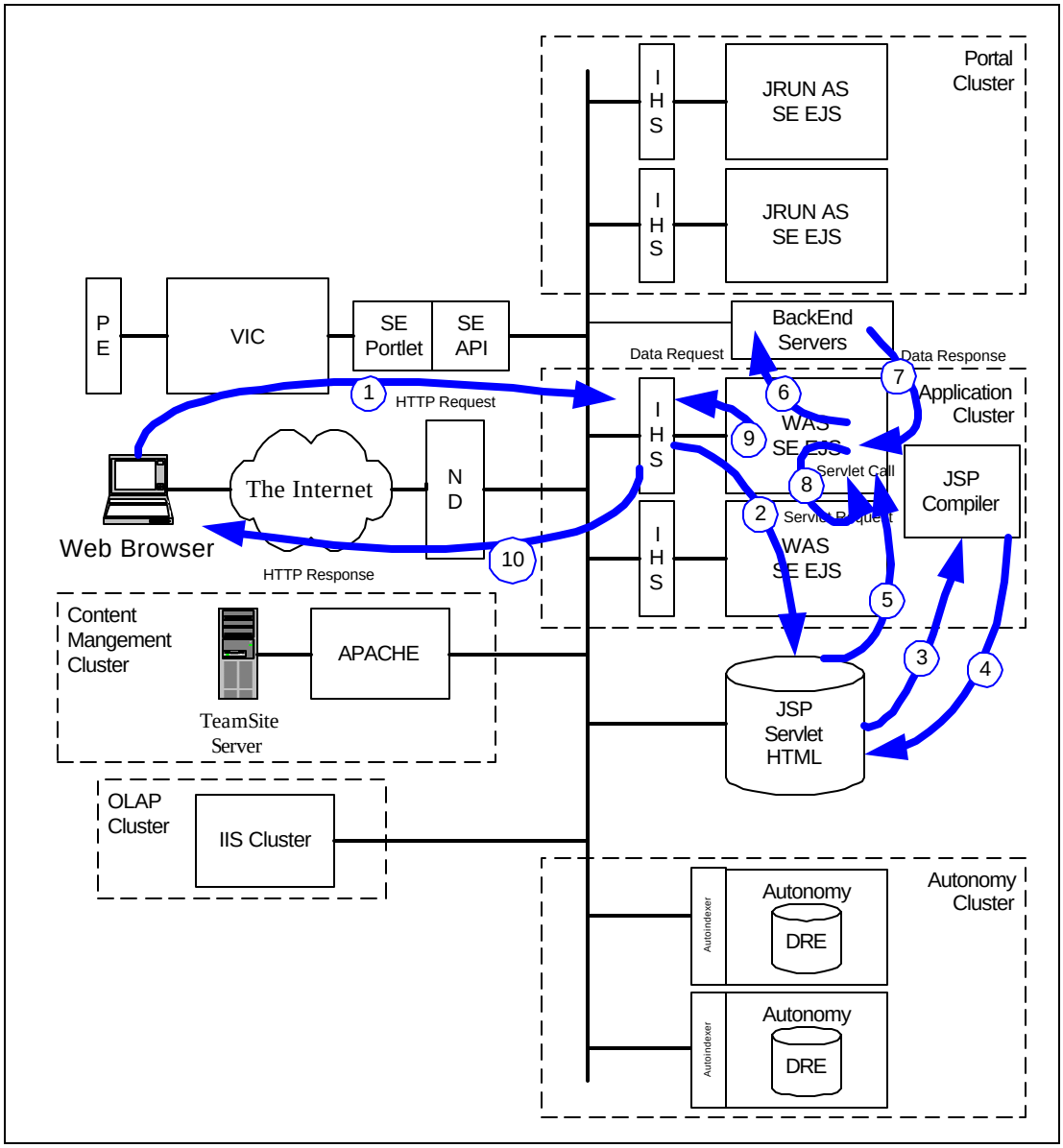

Figure 11 – Touch Point # 5 – Detail Diagram: JSP Flows Within WebSphere Application Server (WAS)

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 23 – TouchPoint # 5 – Step by Step Description

Step	Description
Step 1	The client Web Browser sends an HTTP Request to the Web Server.
Step 2	The Web Server sends the JSP Request to the JSP Repository.
Step 3	The JSP repository sends the JSP to the JSP Compiler.
Step 4	The JSP Compiler produces a servlet and then it sends it to the servlet repository.
Step 5	The Web Server sends a servlet request to the servlet repository and the servlet engine processes the servlet.
Step 6	A data request is then sent to the backend server.
Step 7	The backend server sends the requested data.
Step 8	The servlet engine continues to process the servlet.
Step 9	The servlet produces HTML that is returned to the Web Server.
Step 10	The Web Server returns the HTML to the client Web Browser.

5.2.6. Touch-Point # 6 (IHS – WAS EJB Interaction)

In this scenario, a servlet requests an EJB service. The Web Server EJS loads the requested EJB witin an EJB Container. The EJB accesses Backend Servers. The requesting servlet receives the EJB response and returns appropriate results to the client Web Browser.

The following diagram illustrates the information flow for this scenario.

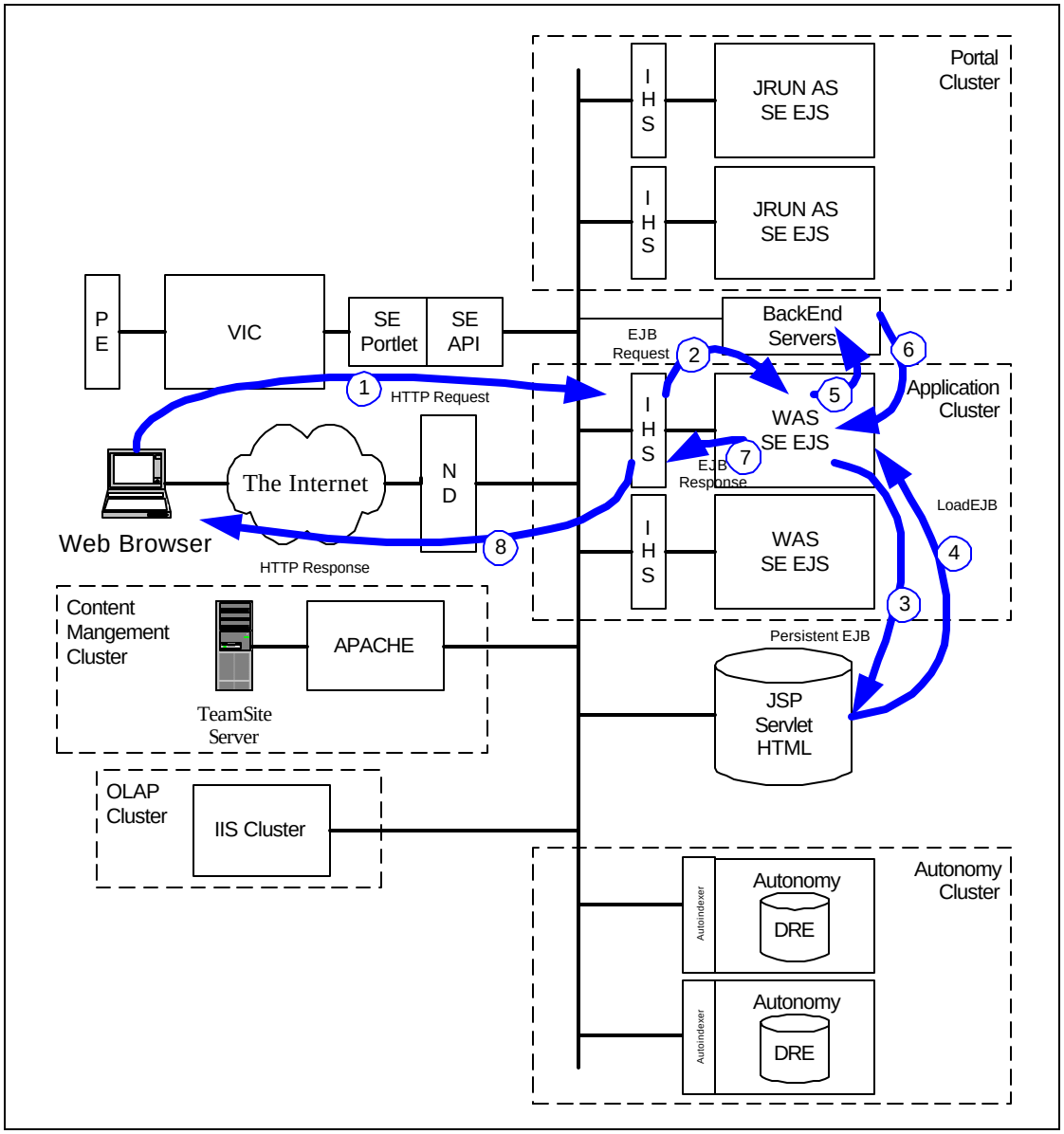

Figure 12 – Touch Point # 6 – Detail Diagram: EJB Interaction

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 24 – Touch Point # 6 – Step by Step Description

Step	Description
Step 1	The client Web Browser evokes a servlet through an HTTP Request..
Step 2	The servlet performs an EJB Request.
Step 3	The EJB Container selects a specific EJB via the Database.
Step 4	The Database provides the selected EJB to the EJB Container.
Step 5	The EJB Container issues a data request to a backend server.
Step 6	the backend server providses the data response to the EJB Container.
Step 7	The EJB Container returns the EJB Response to the servlet.
Step 8	The servlet provides the HTTP Response to the requesting client Web Browser.

5.2.7. Touch-Point # 7 (Autonomy – Document Content Publishing)

OpenDeploy will manage documents to their final production output. Once the document becomes production, OpenDeploy will deposit the file out to a directory structure on the production SAN.

The Autonomy Autoindexer process polls the SAN directory for new documents to index. It then imports the data into an IDX file and sends a command through HTTP to the DRE. The DRE is given the name of the IDX file to index. The DRE then locates the file and indexes its contents into the DRE database.

An IDX file will contain information that has been formatted in Autonomy's import format, so that it is ready to be indexed into the DRE. A typical IDX file entry will look like this:

```
#DREREFERENCE http:///www.mysite.com/us/DailyNews/teenviolence0325.html
#DRETITLE Why Do Teens Kill?
#DREFIELD Summary="Explaning Acts of Brutality by Youngsters Why Do Teens Kill?"
#DRESECTION 0
#DREDATE 925668001
#DREDBNAME Database
#DRESTORECONTENT y
#DRECONTENT
Explaining Acts of Brutality by Youngsters Why Do Teens Kill? Dr. Howard Spivak,
\chirman of AAP Task Force on Violence, comments on kids using violence to
\solve problems Luke Woodman is accused of killing his mother, then going to
\school and shooting nine students in Pearl, Miss.
\[…]
\content here
\[…]
\As a very common theme. In 80% of the cases it has been the case.
#DREENDDOC
```

The following table lists a point-by-point breakdown of the above IDX file entry.

Table 25 – Touch Point # 7 – Listing of IDX File Entries

Key	Value
#DREREFERENCE	Original URL for this page.
#DRETITLE	Document title.
#DREFIELD *Summary*	Short summary of the document.
#DRESECTION *Name*	User-defined fixed and variable fields.
#DREDAT *nnnnnnnnn*	Document date in epoch seconds.
#DREDBNAME *Databasename*	Name of the DRE database into which this content is to be indexed.
#DRESTORECONTENT (*y* / *n*)	Flag telling the DRE whether or not it should store this content.

Key	Value
#DRECONTENT	#DRECONTENT is followed by a newline character, then the actual content to be indexed for this record. Content is bounded by a newline character, or by the next #DRE key encountered.
#DREENDDOC	Indicates the end of this record.

The following diagram illustrates the information flow for this scenario.

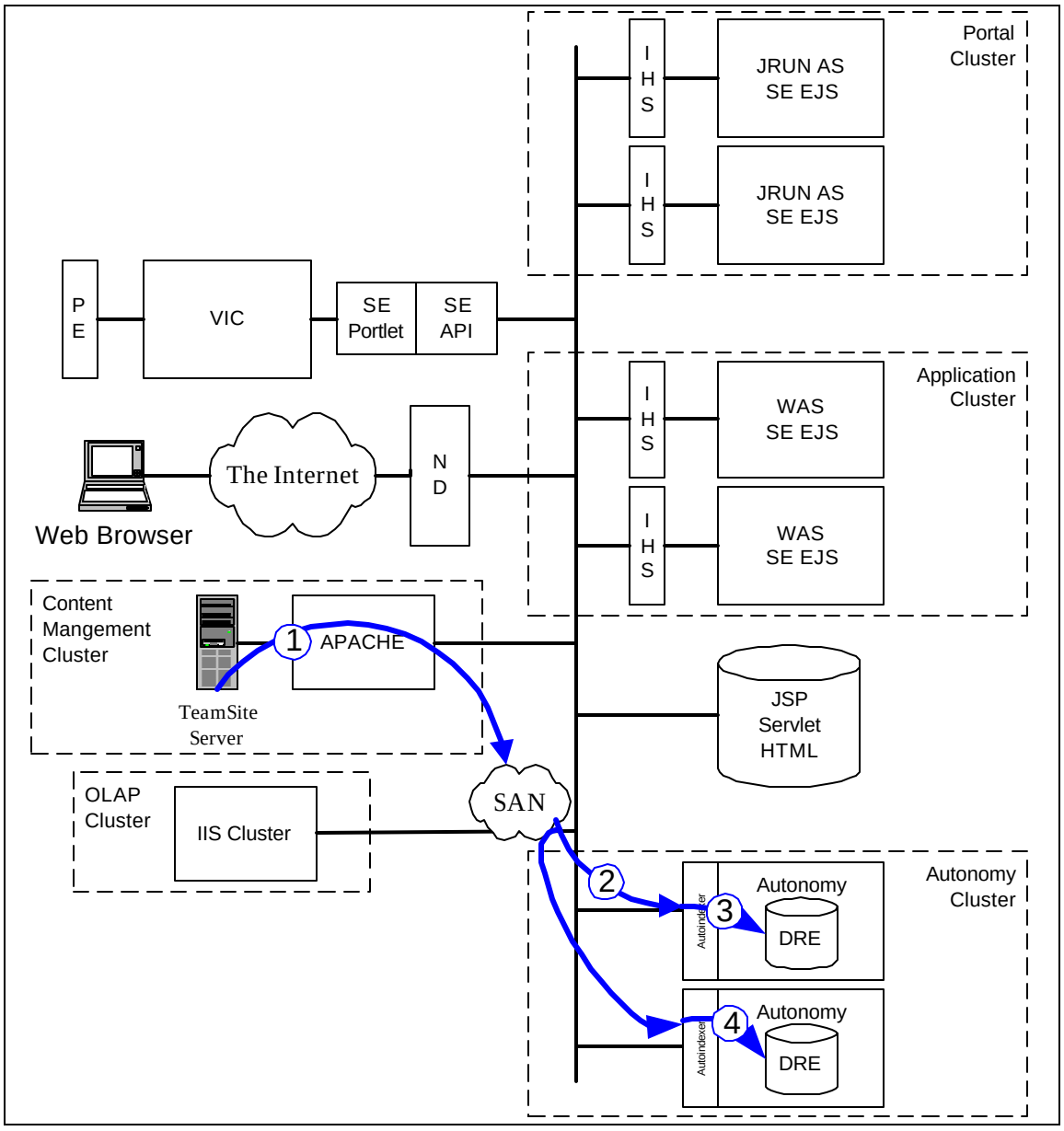

Figure 13 – Touch Point # 7 – Detail Diagram: Document Publishing Indexing Process Flow

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 26 – Touch Point # 7 – Step by Step Description

Step	Description
Step 1	OpenDeploy pushes content to production SAN to a location defined as ../autonomyidx.
Step 2	Files from a directory structure are taken by the Autoindexer and an IDX file is created on the SAN.
Step 3	The Autoindexer 1 signals the DRE to read the IDX file on the SAN and indexes the IDX into DRE 1.
Step 4	The Autoindexer 2 signals the DRE to read the IDX file on the SAN and indexes the IDX into DRE 2.

5.2.8. Touch-Point # 8 (Autonomy – URL Content Indexing)

The Autonomy WebSpider uses HTTP request to gather documents from a URL site. Once the spider process has obtained a document it automatically analyzes the document for links to other documents. These links are followed conditionally based on the setting specified in the spider configuration file; further documents may be retrieved. The document retrieved are then automatically imported into the Autonomy indexing file format and created on the SAN. OpenDeploy then keeps track of different versions of the index file and recreates the index file on the SAN. The Autonomy AutoIndexer process then indexes the index file into the Autonomy DRE database.

The process is duplicated with AutoIndexer 2 indexing the same index file into the Autonomy DRE2 database.

The following diagram illustrates the information flow for this scenario.

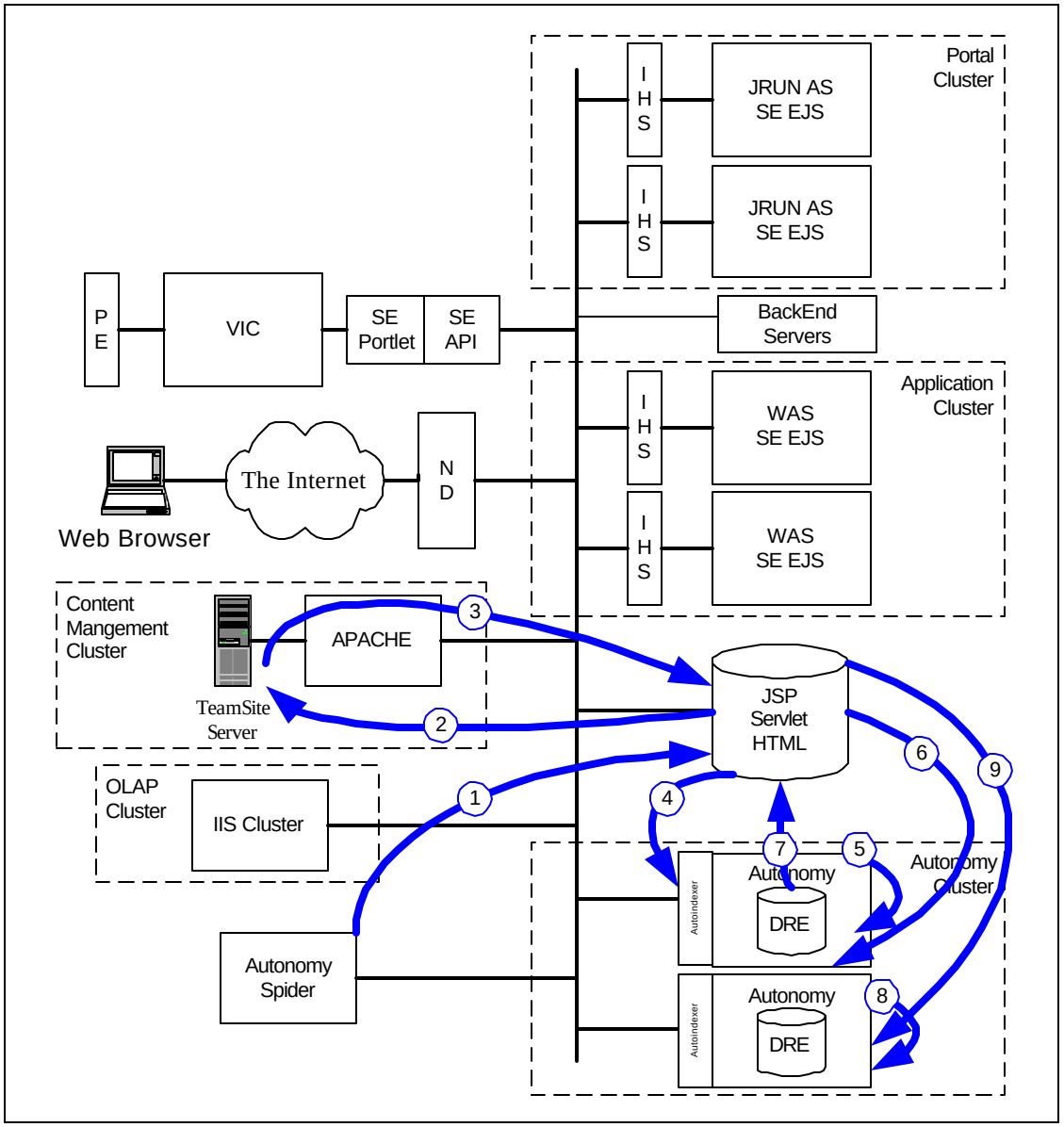

Figure 14 – Touch Point # 8 – Detail Diagram: Indexing of URL Content

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 27 – Touch Point # 8 – Step by Step Description

Step	Description
Step 1	The Autonomy spider can run multiple jobs. Each job represents a URL that is to be spidered. The output from the spider is an Autonomy IDX file. The IX file has a .idx extension and is created on a partition of the SAN in a directory called ../interwovenidx.
Step 2 & 3	OpenDeploy imports the Autonomy index file from the SAN. OpenDeploy will recreate the Autonomy index file on the SAN in a directory called ../autonomyidx.

Step	Description
Step 4 & 5	The Autonomy AutoIndexer process polls the ../autonomyidx directory for Autonomy index files. If the AutoIndexer finds an Autonomy index file to process it sends a message to the Autonomy DRE with the name and location of the index file. The DRE then indexes the index file into the DRE database.
Step 6 & 7	After the DRE indexes the index file successfully the index file is moved to another directory called ../autonomyidx2 which is being polled by the AutoIndexer2 process.
Step 8	When the AutoIndexer2 process finds an index file to process it sends a message to the DRE2 passing the location and name of the index file.
Step 9	The DRE2 processes the index file by indexing into the DRE2 database.
Step 10	After the DRE indexes the index file successfully, the index file is deleted off the SAN

5.2.9. Touch-Point # 9 (Autonomy – High Availability Configuration)

The high availability Autonomy configuration consists of two DREs. Both DREs will be the primary production search engine. The content of both DREs will be duplicated by having the Autonomy AutoIndexer process index the same information into each of the DREs. The DREs will be configured as a ND Cluster.

When a user submits a query using the Autonomy search API, the domain name of the ND will be passed as a property to the Automony Search API. The ND will determine which Autonomy DRE is active and will load balance the Autonomy API search requests across the two DREs. If a DRE goes down the ND will submit the search API to the remaining active DRE.

The output from the Autonomy search API will be an array of results in string format that needs to be passed by the executing servlet and published out to an HTML table. The HTML table is displayed to the user.

The following diagram illustrates the information flow for this scenario.

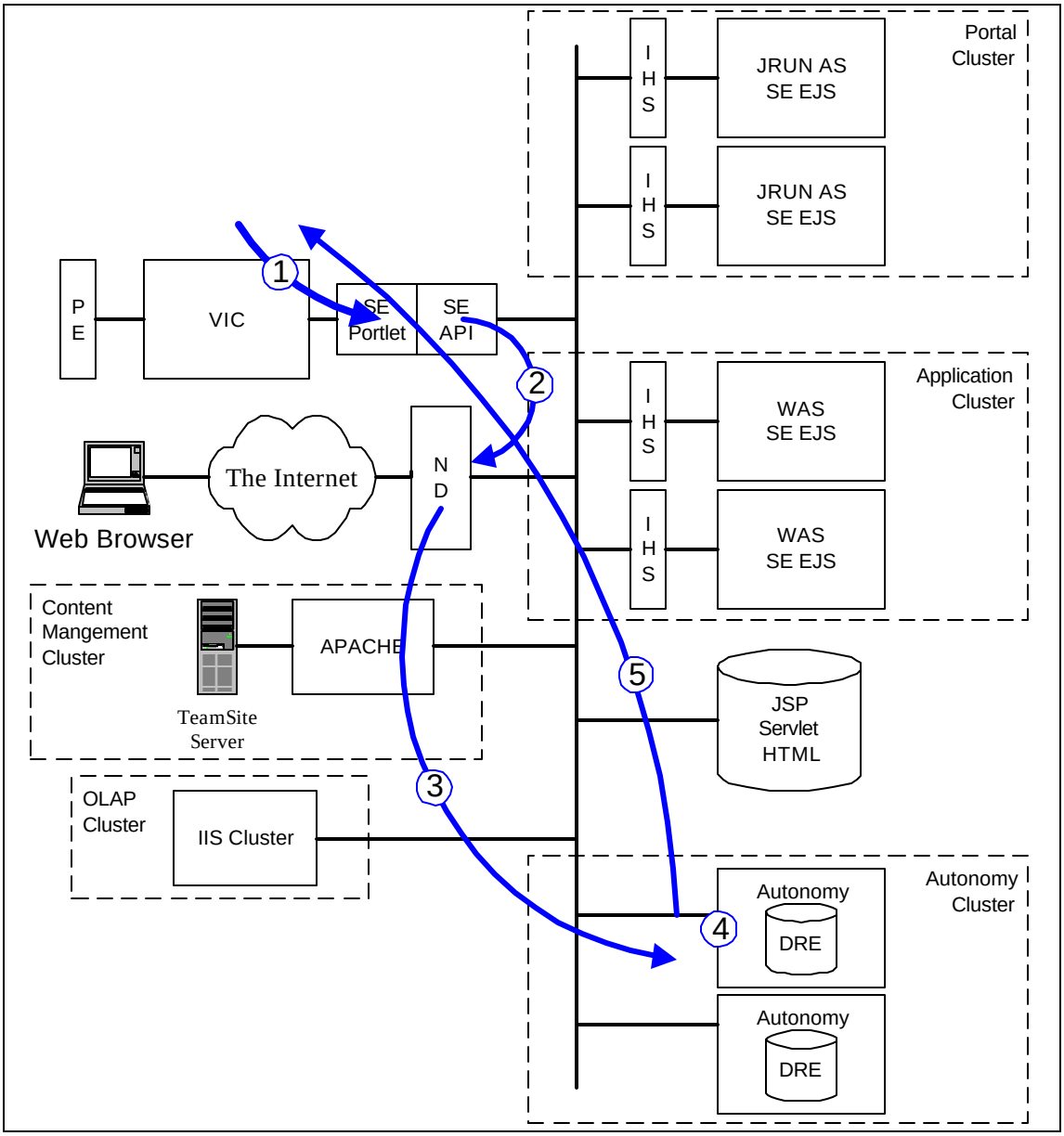

Figure 15 – Touch Point # 9 – Detail Diagram: High Availability Solution

The following table lists discrete steps describing the sequence of events and information flow within the touch-point scenario.

Table 28 – Touch Point # 9 – Step by Step Description

Step	Description
Step 1	A search request is invoked by the user, resulting in the SE portlet formatting the user search argument and search method.
Step 2	The Autonomy KB API formats the user search request into an HTTP request where the host portion is the ND virtual address of the Autonomy cluster.

Step	Description
Step 3	The ND balances the load of the Autonomy cluster and redirects the URL to one of the available Autonomy DREs.
Step 4	The DRE performs the search request.
Step 5	The DRE returns the search results as an HTML text string.
Step 6	The Viador portlet acts as a Web Browser to the Autonomy DRE server.

5.2.10.

5.2.11. Network Dispatcher Topology

The following diagram provides a basic representation of the ND Topology.

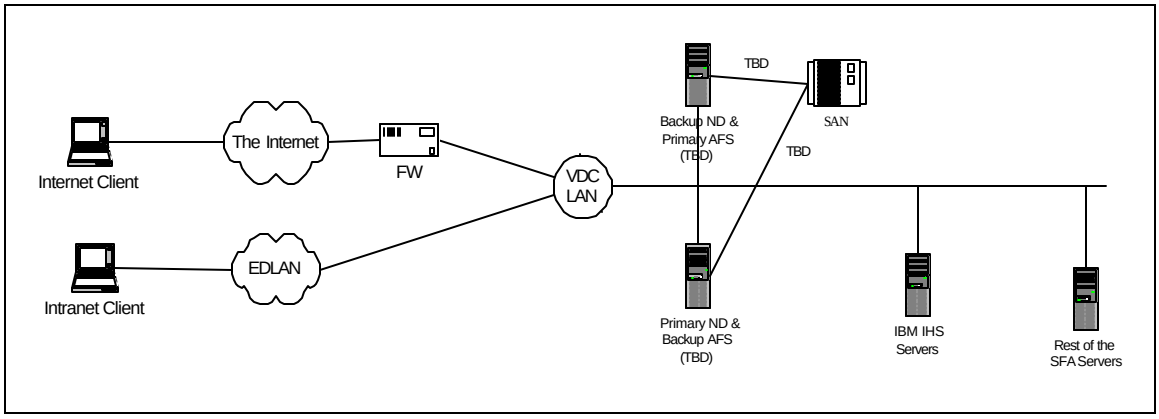

Figure 16 – Network Dispatcher Topology

The ND for the SFA project leverages the capabilities of the Dispatcher and the ISS components. The ISS monitoring capabilities on the TCP servers (IBM HTTP Servers and IBM WebSphere Application Servers) provide the Dispatcher with server load information. In this design, the ISS cooperates with the Dispatcher, but ISS does not make any load balancing decision. The ISS monitor collects specific server information such as CPU usage, memory usage and disk activity from the ISS agents running on the individual servers, and forwards it to the Dispatcher. The Dispatcher uses this load information, along with other sources of information, to determine which is the least loaded server of the cluster and then performs load balancing.

The following diagram depicts the ND system for the SFA environment using the Dispatcher and ISS.

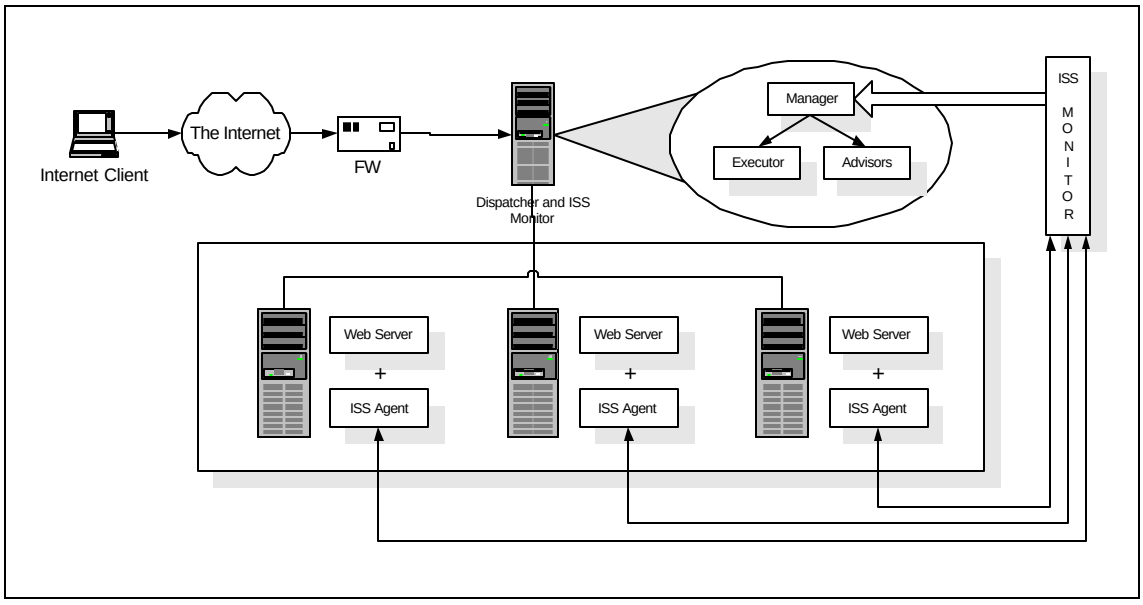

Figure 17 – Network Dispatcher SFA Configuration

Network Dispatch Design Overview

IP Packet Flow

The following diagram is a representation of the IP packet flow in the ND environment.

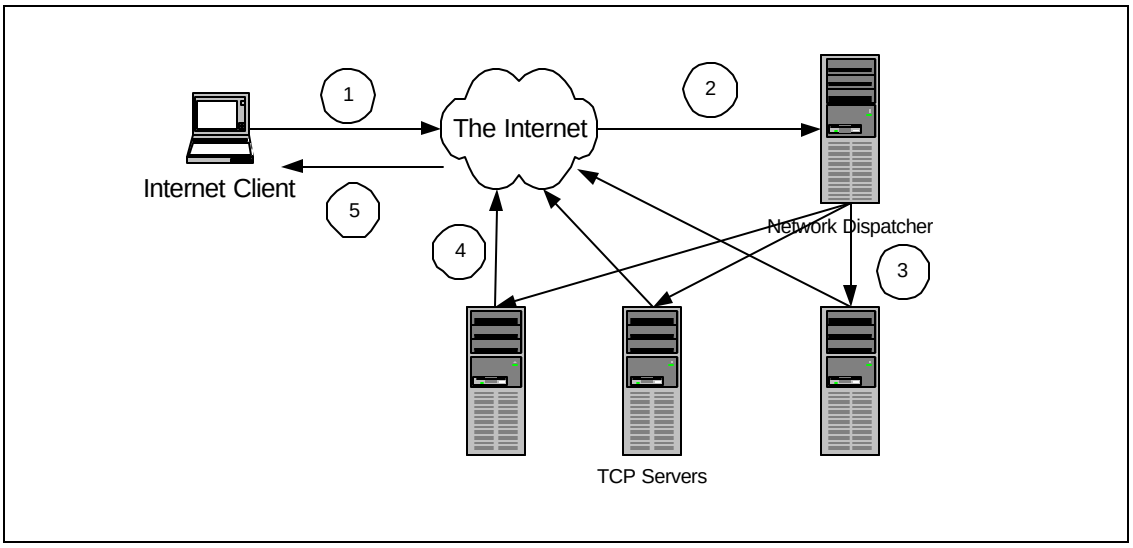

Figure 18 – Network Dispatcher IP Packet Flow

IP configuration must be performed on the Dispatcher servers and on the TCP servers (Web servers). In addition, aliases need to be defined to the cluster address on the network interface on the Dispatcher servers and the loop back devices on all the cluster's TCP servers. The Dispatcher makes several TCP servers appear as one in the TCP / IP environment, typically for HTTP, FTP, and other protocols on the Internet.

In the ND environment incoming IP packets sent by users to the cluster IP address first arrive at the Dispatcher machine, not at one of the TCP servers. This is because the Dispatcher's network interface, besides having its own unique IP address, has been given an alias to the cluster address. The TCP servers in the cluster also have an alias to the cluster address, but this is defined on the loop back interface. The Dispatcher runs at a low level in the server's operating system so it can directly intercept all the IP packets. Each time a new connection is initiated by a client, the Dispatcher selects which TCP server in the cluster should receive the connection packet. Now the Dispatcher should be able to send that packet to the selected TCP server. The Dispatcher routes the packet based on the MAC address of the network adapter on the chosen TCP server.

The original destination MAC address on the IP packet was the one of the network interface on the Dispatcher machine itself. When the Dispatcher selects the MAC address of the selected TCP server, it changes the original destination MAC address on the packet to the selected MAC address. The packet is then encapsulated in the frame and transmitted to the chosen TCP server. The Dispatcher then sets up a connection table entry to make sure that

subsequent incoming IP packets for this client continue to be forwarded to the same TCP server, until the connection is terminated.

When the TCP server receives the packet, the information related to the MAC addresses is eliminated and the source and destination IP addresses are extracted. The destination IP address is still the cluster, but the TCP server has its own IP address. However, the TCP server can accept that packet, since the cluster address is configured as an alias on the TCP server's loop back interface. At this point, the TCP server sends a response to the Web client that originated the request.

Once the TCP server receives all the IP packets of the originating client's request, it performs the standard TCP processing that is commonly performed by all TCP servers while responding to a client. It switches the IP source and destination addresses for the outgoing packets that form the response to the client. The source address becomes in this case the cluster address, while the destination address is now the client IP address. This operation has an important consequence; the balancing function is transparent both to the client and the clustered servers.

The destination IP address is the client's IP address, not the Dispatcher's. For this reason, the TCP server can route the IP packets through its default router directly to the client, and all the outgoing packets do not pass back through the Dispatcher. There is no need to even return using the original physical path and a separate high-bandwidth connection can be used. This is very important, since in many cases, the volume of the outbound server-to-client traffic is substantially greater than the inbound traffic. HTML pages and imbedded images are typically 10 times the size of the client URLs that requested them.

The source IP address in the outgoing packet sent by the TCP server shows as the cluster address, and not as the TCP server's IP address. The cluster address was also the destination IP address in the IP packets sent by the client in its request, so the client is not able to understand the TCP architecture of the target server. This security feature offered by the Dispatcher ensures privacy for a site that is composed of multiple TCP servers load-balanced by a Dispatcher server.

The client and servers must be part of the same Intranet. In this case, the network-monitoring tool would show the MAC address of the TCP server that actually served the request. The MAC address shown in the packet is that of the router closest to the client.

The Dispatcher does not participate in bi-directional communications with the client but simply forwards the incoming packets unchanged, its presence is transparent to both client and server. The real TCP / IP connection is between the client and the clustered server, and the Dispatcher soon disappears from the scene after forwarding the incoming packets. The only requirement for the TCP server is that its loop-back device be set or aliased to the cluster address. In this way, the server is capable of responding to a request that was addressed to the cluster address.

ND Servers

The ND servers consist of IBM Sun Microsystems Enterprise servers. These are the same servers used for the AFS. The primary server used for the AFS is the backup server for ND and the backup server used for AFS is the primary for ND. The ND servers will be classified as Database Server, File Server, System Control Machine, and Binary Distribution Machine. The responsibilities of the ND primary server include:

- Executor – This function supports port-based routing.
- Manager – This function sets weights used by the Executor based on the feedback from the ISS monitors and the Executor.
- Advisor – This function sends requests to TCP servers to measure actual client response time for a particular protocol and feeds such information to the Manager.
- ISS Monitor – This function provides the feedback to the Manager through the ISS agents.

The responsibilities of the ND backup server include the same functionality as the primary server. However, the ND backup server will only be used upon the failure of the primary ND server.

High Availability Design

ND is designed for high availability. A standby Dispatcher server, the backup server, remains ready at all times to take over load balancing should the primary Dispatcher fail. The high-availability configuration detects and recovers from network and server failures. The Dispatcher is able to determine that a server or a network is down. In case of failure, clients lose only the current connections, but they can immediately establish a new connection to the remaining servers with no problems.

The high-availability environment involves the two Dispatcher servers, the primary and backup, with connectivity to the same clients, and to the same cluster of servers, as well as connectivity between the Dispatchers.

The primary machine works normally as a Dispatcher, and is in the active state while it is load balancing the TCP servers of its clusters. The backup machine configured in a very similar way to the primary machine, stays in standby mode unless the primary fails. The two machines are synchronized, and only the primary machine routes packets, while the backup machine is continually updated.

The two machines establish communication to monitor the status of each other, referred to as a heartbeat, using a specific port. If the primary machine fails, the backup machine detects this failure, switches to active state, and begins to take over the routing of packets. When the primary machine is operational again, but in standby state, depending on the configuration, it either automatically becomes the active machine, or stays in standby mode. In standby mode, a manual intervention is required to make it active again.

ND Clients

The ND system, in the SFA environment, has the TCP servers as its clients. These clients are the IBM IHS Servers.

Networking and Interfaces

The current design of the ND servers for ITA relies on the SFA VDC LAN environment for connectivity between:

- The ND servers and the IBM IHS servers.
- The ND servers and the Web browser clients through the firewall.
- The ND system in the SFA environment for the initial release will not operate over the WAN.

Local Area Network (LAN)

The network topology required for the ND system is Ethernet. TCP and UDP are the communications protocol used between the clients and the servers. The LAN connection will support the following:

- Connectivity to the IBM IHS server
- Connectivity to the Internet
- Connectivity to the Intranet

Cluster Address and Non-forwarding Address

Two ND Clusters will be defined for the SFA project using the same network interface card. One cluster for the Internet access and a cluster for the Intranet access. Therefore, two aliases will be configured to match the two clusters. The Dispatcher server will require four IP addresses as described below.

Two primary IP addresses are required for the primary and backup Dispatcher servers. This type of IP address is also known as the non-forwarding address, is the IP address of the hostname. The cluster address is the IP address that will be used by clients to access the entire site. The Dispatcher' will be dedicated to load balancing requests that are sent to this address. A hostname can be associated to this cluster address.

A cluster IP address is required for the Internet network. This is a unique IP address by which client requests access the cluster. It is a virtual address that is valid only locally. Therefore, no other server on the network should have the same IP address as the cluster address. The same IP address, in the production environment will be used for the primary and backup clusters.

A cluster IP address for the Intranet. This also is a unique IP address. The same IP address, in the production environment, will be used for the primary and backup clusters.

Wide Area Network (WAN)

The WAN is not required for the initial release.

TCP and UDP Ports

The ND system can load balance any TCP or stateless UDP application. However, these ports need to be defined for each protocol for the ND.

The following table lists the ports required for the initial release.

Table 29 – Protocol Definition for ND System Load Balancing

Protocol	Port
FTP	20
FTP Control Port	21
SSL	443

Advisor protocols and ports

In order to feed the Manager with information about the status of the IBM IHS Servers, the Advisor must be configured and started. The following is a table of available Advisors along with their respective protocols and ports.

Table 30 – Advisor Protocols and Ports

Advisor Name	Protocol	Port
ftp	FTP	21
Telnet	Telnet	23
Smtp	SMTP	25
http	HTTP	80
Pop3	POP3	110
nntp	NNTP	119
SSL	SSL	443
Workload Manager	Private	10,007
WTE	HTTP	80
PING	ping	0

5.3. Database Server

The database component provides a persistent data storage and retrieval service organized as an Operational Data Store (ODS). The preferred access method for the ODS is JDBC for Java applications; and Open Database Connectivity (ODBC) for non-Java applications.

Native Oracle drivers are also supported for specific application or other COTS products. The recommended access approach is to consolidate application accesses into a few calls using Stored Procedure Calls.

The Database component will use ORACLE 8i and will be installed in a redundant Hewlett Packard (HP) N-class environment. Database replication is utilized to achieve high-availability.

The information managed by the database server component is either application or COTS product specific. The databases associated with applications are not addressed as an integral part of the IA. The databases associated with the IA COTS products are specified and discussed with each product.

5.4. Component Integration

The search engine can be invoked by the Portal component. The following diagram illustrates how the search engine interacts with the Portal component.

5.4.1. Viador Autonomy Portlet Integration

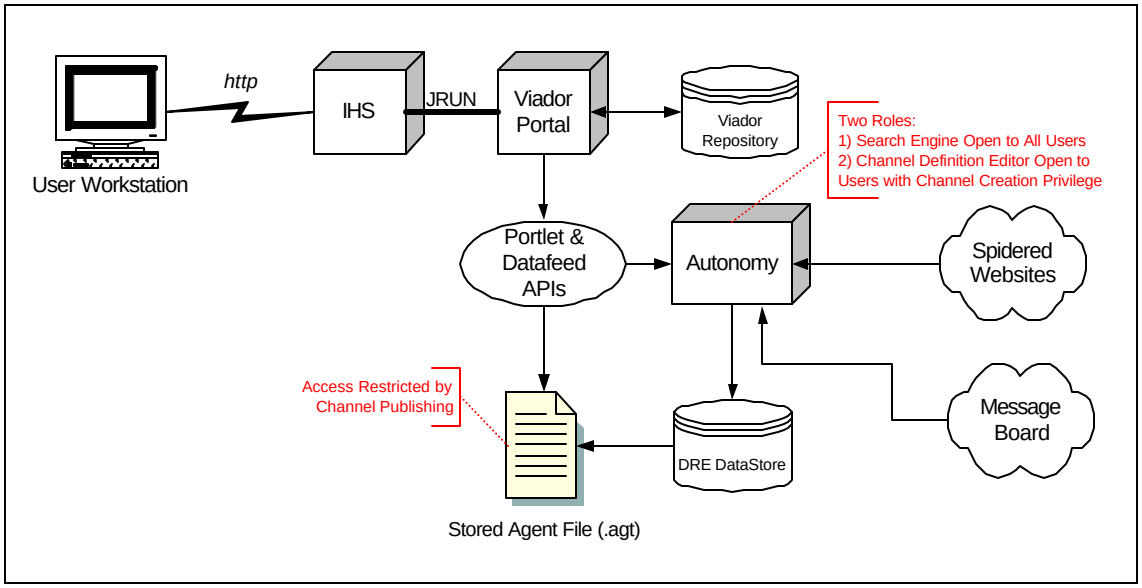

Figure 19 – Viador Autonomy Portlet Integratoin

The Knowledge Management component integration with the Portal component has been identified in the following parts:

Search/Advanced Search

Search engine components will use the Portal component Data Feed API using search field and advanced search functions that are abstracted by the Portal component. The search field

will allow search users to enter natural language information and exploit the search engine's searching rules as well as other advanced search capabilities.

Categories/Communities

Integration of search engine agent technology will be used to categorize information and to auto-feed channels and / or folders in the portal. These agents will be registered as object types (links within the portal) that define and populate channel information; in particular this would be a channel stated as relevant information. These links would point to the search engine agent and execute them upon request.

Collaboration

The search engine has the capability to collaborate this agent technology by sharing with other users similar types of research. The Portal component Data Feed API filters and displays, using embedded security, the files to which each user has authorization. Collaboration of the search engine agents will need to be defined as a separate portlet within the portal itself, and needs to display the other users that are performing common searches.

5.4.2. Interfaces With Other Applications

Search Engine Application Server Integration

The Autonomy Search Engine will work with any Web Server that supports the following:

- JSP
- CGI Scripts
- POP3

The configuration of the search engine and a Web server is straight forward since the search engine only uses the Web server to present its user interface. The configuration involves locating the Web server HTML root directory and copying all of the search engine HTML files. Another parameter is locating the Web server CGI script directory and copying all of the search engine's CGI scripts.

5.4.3. Search Engine Portal Integration

The integration of the search engine and the portal requires the Java Toolkit supplied by the search engine and using the Data Feed API supplied by the portal.

The search function allows users to enter natural language information to exploit the search engine's search rules as well use the search engine's advanced searching capabilities.

The following diagram illustrates the search engine portlet.

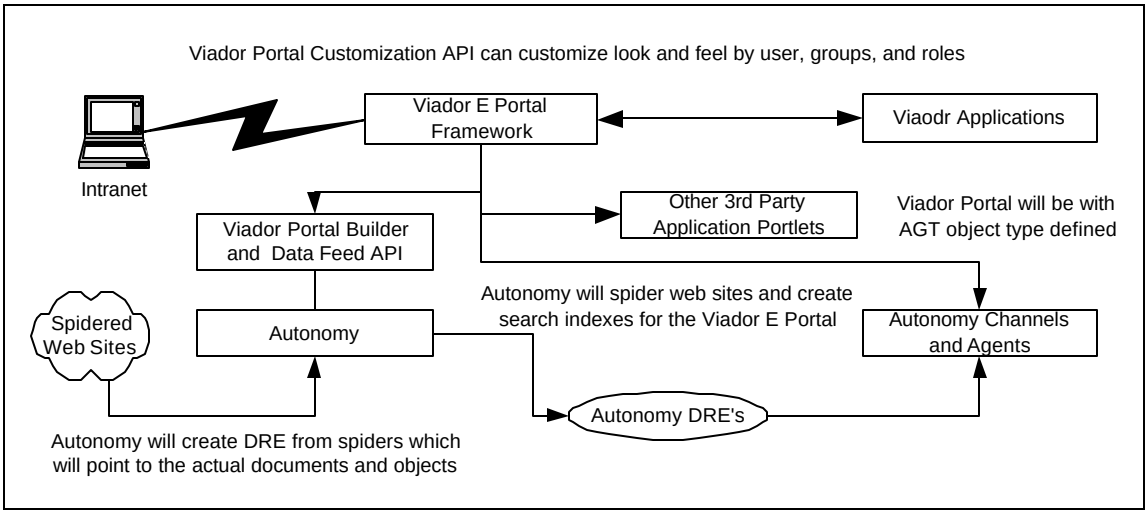

Figure 20 – Search Engine Portlet

5.4.4. Search Engine with the Content Management Tool

The integration of InterWoven's Content Management and Autonomy's Search Engine is accomplished with out of the box capabilities. Autonomy will not index any InterWoven content until the content has reached its final output. The content will be copied to a shared directory where the Autonomy AutoIndexer process will monitor and index any new content that is published to the shared directory.

5.4.5. Message Board

The Autonomy Search Engine has the ability to provide white-board messaging. The Autonomy Message Board will be defined as a separate Viador portlet and will be available to users through the security of role assignments and applications rights.

6 Performance Considerations

The following sections discuses the specific IA design for performance considerations. The performance of the IA is based on the performance of the architecture components and the VDC environment that supports the IA servers. The IA design replicates certain components to insure that adequate capacity is available.

6.1. Web Browser

The individual user provides the Web Browser component and adequate performance is assumed. Specific application requirements may levy performance or capacity requirements on the client.

6.2. Firewall

The Firewall component is provided by the VDC and adequate performance is assumed. The anticipated client connection request volume is expected to be within the capacity of already installed Firewall components.

6.3. Load Balancing

The ND Manager decides which is the least-loaded server on a particular port in the cluster by looking at the weight of each server. The Manager will periodically update the weight of each of the server machines, basing its decision on four parameters or policies:

- The number of active connections on each TCP server
- The number of new connections for each TCP server
- Input from TCP server Advisors
- Information from system monitoring tools, such as ISS

Using the method called proportions of importance performs setting the servers' weights in the load-balancing process. Each of the above factors is attributed a number, from 0 to 100, that acts as a percentage. 0 means that the policy is not used while 100 means that only that factor will be used. It is necessary that those proportions add up to 100. The default settings at the startup appears as 50 50 0 0.

6.3.1. Guidelines on Proportions of Importance Settings

Although there are generally no fixed rules on how to set the proportion value, it is still possible to provide some useful guidelines.

The first two proportions are related to active and new connections respectively. Typically in an out-of-the-box configuration, this is all you will be able to use. That is why the default proportions at startup of ND appear as 50 50 0 0.

The load on a classical Web server depends mainly on the number of connections, both active and new. If the client connections to the services provided by the TCP server machines are quick (such as small Web pages served using the HTTP GET method), then the number of active connections will be expected to be fairly low. On the contrary, if the client connections are slower (such as database queries), then the number of active connections will be higher.

If the Manager started, the value for the Advisor proportion may need to be non-zero. The standard Advisors shipped with ND execute a trivial transaction on each TCP server. Experience shows that it is not a good idea to set the Advisor proportion to a high value. We recommend 49 49 2 0 for this setup.

If the ISS daemon is installed on the servers, the ISS proportion needs to be added.

6.4. Web Server

Based on application requirements.

6.5. Application Server

Based on application requirements.

6.6. Component Broker

Based on application requirements.

6.7. Content Management

6.7.1. Performance

TeamSite does not have any kernel tuning parameters that need to be configured. The only tuning would be what is required of the Web Server component. The following sections describe the essential TeamSite performance configuration settings.

Cache Size

To set the TeamSite cache size, edit the cachesize line in the [iwserver] section of iw.cfg. If a comment symbol (#) is present at the beginning of this line, remove it. If this line does not appear in the iw.cfg file, add it as shown below. The initial cache size setting should be approximately three times the number of files and directories on the largest branch. For example, if the largest branch contains 15,000 files and directories, you should set cache size to 45000 as shown below.

The minimum cache size is 1000; maximum is 500000 (five hundred thousand). Each cache line takes a maximum of 1KB of physical memory. Recommended physical memory is cache size times 1KB plus an additional 25% as a safety margin. In the example shown below,

physical memory would be (45,000 * 1KB) + 11MB = 56MB. If there is a great deal of memory swapping, either reduce the cache size or install more memory.

The TeamSite server must be restarted for these changes to take effect.

cachesize=45000

RPC Threadcount

The RPC threadcount setting determines how many simultaneous requests TeamSite can handle from users via the GUI or command-line tools. These requests are very short-lived, so that threads are quickly freed for other users. If all threads are currently being used, TeamSite starts to serialize requests. This setting should not be altered.

```
rpc_threadcount=64
```

File System Threadcount

The file system threadcount should be set to approximately the number of CPUs on the TeamSite server. This setting should not be set higher than 2. To change the file system threadcount, edit the fs_threadcount line in the [iwserver] section of iw.cfg. If a comment symbol (#) is present at the beginning of this line, remove it. If this line does not appear in your iw.cfg file, add it as shown below.

The TeamSite server must be restarted for these changes to take effect.

```
fs_threadcount=1
```

Filesystem Active Area Cache

The file system active area cache should be set to approximately the number of users who are expected to be using TeamSite concurrently. This is the number of users who are using TeamSite at one time, not the total number of TeamSite users. If this value is too large, it will significantly impact memory usage.

To set the file system active area cache, edit the fs_active_area_cache line in the [iwserver] section of iw.cfg. If a comment symbol (#) is present at the beginning of this line, remove it. If this line does not appear in your iw.cfg file, add it as follows:

The TeamSite server must be restarted for these changes to take effect.

```
fs_active_area_cache=8
```

6.8. Portal

The Portal component provides the ability to support thousands of concurrent portal sessions while delivering consistent response time to incoming requests from portal users. To do so requires at least 300 MB of physical disk space and 256 MB of memory available on the server

machine in order to satisfy the needs of 10 user's licenses. An additional 2 to 4 MB of memory per each concurrent user license is also recommended.

6.9. Knowledge Management

The search engine is CPU intensive and operates from a central location, thereby increasing the demands of the network. The size of the request and through-put could possibly place a strain on the network on which it resides. This also impacts both the insertion and retrieval of information.

One solution to improving performance is creating additional instances of the DRE. As a rule, adding more processors and memory will increase the search engine's system performance. In addition, creating multiple instances of the DRE on one physical machine can also improve performance.

A reasonable document threshold is 1.5 million documents to each DRE instance.

6.9.1. Operational Monitoring

The Autonomy administrator tool is used to monitor both query responses and load indexing. The operational monitoring tool reads the search engine log files that are provided with the product.

The DRE process can be monitored by issuing an HTTP command from any web browser. The URL is http://dreipaddress:drequeryportnumber/qmethod=v. This command will provide information such as the number of documents resident in a DRE, the query response times, a listing of all query and indexing requests, information about every HTML file that the spider job touched, whether an HTML file was indexed into the DRE, and if not, an (English) explanation why the HTML file was not indexed.

An alternative path to finding all DRE activity is to look in the DRE working directory, at the file called 'queryh.log'.

6.10. Directory Server

The Directory Server component is provided by the VDC and adequate performance is assumed. Specific application requirements may levy specific performance or capacity requirements on the VDC implementation.

6.11. File Storage

6.11.1. AFS Client Tuning

AFS offers a number of parameters for tuning client performance and behavior. The following parameters affect performance:

- The Cache Size determines how much AFS data can be cached locally on the client. The obvious consideration in determining the cache size is the amount of available disk space on the client, but the cache size should not be made extraordinarily large. The number of users on the machine and the type of data being accessed should be considered when determining cache size. A Windows NT workstation being used for office applications would want to have a cache no larger than 30-60 MB.

- The Chunk Size determines the size of the data chunks sent from the File Server. A small chunk size requires more memory usage by the client and may lead to performance problems, but a larger chunk size may waste cache space if a lot of the cached files are smaller than the chunk size setting. Typically this setting is adjusted upward for both large data files and fast network technology.

- The Status Cache setting determines the number of cached files about which the Cache Manager retains status information. Adjust this setting upward if the client is caching many small files as opposed to fewer large files.

- The Probe Interval sets how often the client contacts the File Servers to make sure they are still responding.

- The Background Threads handle background tasks for the Cache Manager.

- The Service Threads handle all initial requests for data; applications could possibly be delayed while waiting for a service thread to complete.

With Solaris, tuning AFS client performance can either be performed via the AFS Client Configuration Control Panel or the Command Prompt. The first consideration is the AFS client cache itself, as it can be used as either a disk-based cache or, if sufficient Random-access Memory (RAM) is available, as a memory-based cache. Also the AFS client cache on UNIX systems can typically be set larger than what Windows NT can accommodate, and for cases of large file and data set access, cache sizes of 500MB to 1 GB or more are possible. Moreover, the Cache Manager process - afsd - accepts a number of different arguments that influence its behavior and performance. Some of these include:

- Memcache initializes a memory cache rather than a disk cache.

- Blocks is the number of 1024-byte blocks in the cache and so overrides the cache size specified in the /usr/vice/etc/cache info file.

- Files sets the number of V-files created in the cache directory, which overrides the default that is calculated automatically by afsd.

Upon startup afsd will set the number of V-files used in the cache depending on whether a memory cache or disk cache is used. If a memory cache is used the number of chunks (V-files) will simply be the cache size divided by the chunk size. For a disk cache the number of chunks will be the largest of the following:

- 100

- (cachesize / chunksize)x 1.5
- cachesize / 10,240
- Stat is the number of entries stored in memory for recording status information about files in the cache; the default is 300.
- Daemons sets the number of background daemons to run on the client. These daemons improve performance by doing pre-fetching and background writing of saved data. The default value is 2, which is adequate for a machine with approximately 5 concurrent users. Values greater than 6 are not generally more effective than 6.
- Chunksize sets the size of chunks (or V-files) in the client cache, expressed as a power of 2; the default is 16 for disk cache (2 16 bytes = 64 KB) and 13 for RAM cache (2 13 bytes = 8 KB). This will also be the size of the data chunks that the client requests from File Servers.
- Dcache sets the number of dcache entries in memory, which are used to store information about cache chunks. For a disk cache this overrides the default, which is half the number of V-files; its use is not recommended for memory caches
- Volumes is the number of memory structures allocated for storing volume location information; the default is 50.
- Biods sets the number of virtual memory (VM) daemons dedicated to performing I/O operations on machines running AIX with VM integration.

Changing or setting any one of these parameters on an AFS client is liable to affect performance, so care must be taken to ensure proper performance monitoring and evaluation.

Setting the Client Cache Size

AFS Client Tuning AFS offers a number of parameters for tuning client performance and behavior such as server preferences. The size of the AFS client cache is probably the single most important factor in determining the performance of AFS. Factors that determine the appropriate cache size for a given client include:

- The number of users working on the machine
- The size of the files with which they usually work
- The number of processes that usually run on the machine

The higher the demand from these factors, the larger the cache needs to be to maintain good performance. Disk caches smaller than 10 MB do not generally perform well. Machines serving multiple users usually perform better with a cache of at least 60 to 70 MB. The point at which enlarging the cache further does not really improve performance depends on the

factors mentioned above and is difficult to predict. An AFS client cache can initially be set at installation and then later adjusted dynamically to near the maximum partition size.

6.12. Database Server

The Database Server component is provided by the VDC and adequate performance is assumed. Specific application requirements may levy specific performance or capacity requirements on the VDC implementation.

7 Configuration Standards/Requirements

7.1. Web Browser

The Web Browser component is configured to disable dynamic HTML.

7.2. Firewall

The Firewall component is configured by the VDC.

7.3. Load Balancing

The system administrator must have root access in order to perform any administrative tasks on the Network Dispatcher. The server component is implemented as a Java class, running through the JVM java executable file. The first step in configuring the Dispatcher is to make connection to a host where the Dispatcher is running. The Executor is the Dispatcher component that routes requests to the TCP and UDP servers. It monitors the number of new, active, and finished connections and performs garbage collection of complete or reset connections.

The administrator redefines the non-forwarding IP address or the host name of the Load Balancing component. The Configuration setting section lists a set of parameters for the Executor that can be changed. Some of them have immediate meaning, such as:

- Maximum number of clusters (default is 100)
- Default maximum ports per cluster (default is 8)
- Default maximum servers per port (default is 32)

For a stable running configuration, the administrator may leave the parameter values that appear by default in the Configuration settings section of the Executor Status. The non-forwarding address is set in place of IP address.

In order for the Dispatcher to route packets, each cluster address must have an alias to a network interface card on the Dispatcher server. There are different methods for creating an alias on the Dispatcher server's network interface card to the cluster address. Only one method needs to be used by the administrator to accomplish the task:

- Using system commands ifconfig or ndconfig
- Using configuration scripts to create this alias
- Using the ND GUI or the command line ndcontrol

The Dispatcher can load balance any TCP or stateless UDP application. The administrator may want to define as many ports as the protocols for the Dispatcher server to communicate through.

The system administrator needs to define the TCP servers that need to be included in the cluster for the configured ports.

The main load balancing function is the Manager. The Manager is the function that collects the information from the Advisors about the servers' conditions. Based on this information, it then dynamically adjusts the weights of the single server to reconfigure load distribution during run time.

The Advisor monitor each server defined on the assigned port, and forwards the information about the server's response time and availability to the Manager.

Configuring the TCP Servers (Web Servers)

Before the Dispatcher starts to forward TCP / IP connection requests to the TCP servers, it is necessary to set up the TCP server machines. On each server in the cluster, the administrator must add an alias to the cluster address on the loop back interface. This is the only configuration necessary in the TCP servers in order for to be load balanced by the Dispatcher.

The loop back IP address is usually 127.0.0.1 and is never forwarded as a destination on the network media. The Dispatcher does not change the destination IP address in the TCP / IP packet before forwarding the packet to a TCP server machine. By setting or aliasing the loop back device to the cluster address, the TCP servers will accept packets that were addressed to the cluster address. Before going on with the TCP server's configuration, the administrator must understand the flow of the incoming and outgoing IP packets.

Cluster Addresses and Network Interfaces

It is possible to configure the Dispatcher to manage more than one cluster, whether the Dispatcher server has one or more network interface card. Following are some scenarios:

- If the Dispatcher server has only one network interface adapter, several aliases can be configured to match the number of clusters defined on the same network interface card.
- If the Dispatcher server has two network interface cards, the administrator can manage two clusters, one on each interface card. Each cluster address must have an alias on the corresponding network interface card.

Managing TCP Ports Used by the Dispatcher

The Dispatcher uses three TCP ports for its communications:

- Port 10099 - This port is used for receiving commands from the ndcontrol program
- Port 10005 - This port is used to receive information from an SDA agent
- Port 10004 - This port is used to receive metric response from ISS

On occasions, the administrator may need to change the TCP port:

- If another application is already using port 10099 or port 10005 for its communication, then the ndserver script needs to be modified to use different ports. The script is located by default under the /bin directory.
- To change the port used for receiving ndcontrol command, change the ND_RMIPORT variable to a new value.
- To change the port used to communicate to an SDA agent, the ND_AFFINITY_PORT variable to the new port number needs to be changed.
- To change the port used to receive metric reports from ISS, the metric_port option must be used when starting the manager. When a metric_port is specified,, a log_file needs also to be specified. In addition, to inform ISS of a change in the metric port number, when the Dispatcher Observer is defined in the ISS configuration file, the new port must also be defined.

7.3.1. Configuring and Managing ISS

The system administrator must have root access in order to perform any administrative tasks on the ISS. These tasks include:

- Starting the ISS Daemon – The ISS will operate in this design as a monitor agent.
- Connecting to a host. The administrator must configure the ISS to make a connection to a host where the ISS daemon is running.
- Adding nodes. The administrator must add nodes to the configuration. The administrator must configure the primary Dispatcher server to be the primary ISS monitor and the TCP servers to be the ISS agents. The administrator must also configure the backup Dispatcher server as the backup ND server.
- Defining resource types. The administrator must add resources such as CPU resources to the configuration.
- Defining Services. The service name must be added to the configuration. In our design, the Service DNS name is not needed since we are using ISS as a monitor to update the Dispatcher.
- Add Node Interface to Service. You add the servers to the defined service.
- Defining Observers. The administrator must select the type of Observer to add. For SFA, it is the Dispatcher.

The administrator can dynamically reconfigure ISS by using the isscontrol command or the GUI.

7.4. Web Server

Based on application requirements.

7.5. Application Server

Based on application requirements.

7.6. Component Broker

Based on application requirements.

7.7. Content Management

The Web Server component is required to access the TeamSite GUI, as well as any Templates that are written for the SFA. The aliases that are needed in order to log into the TeamSite product are specified at product installation.

The DataDeploy component requires the ODS instance server name, database name, connection port, and user name and password to populate tables in the ODS.

7.7.1. TeamSite Server Configuration Requirements

The following table identifies the recommended physical cache memory for the TeamSite server.

Table 31 – TeamSite Server Cache

Total Users	Files Per Branch		
	0 to 25,000	25,000 to 75,000	75,000 +
0 to 25	• 1 CPU • 256 MB Memory	• 1 CPU • 512 MB Memory	• 2 CPUs • 1 GB Memory
25 to 50	• 2 CPUs • 256 MB Memory	• 2 CPUs • 512 MB Memory	• 2 CPUs • 1 GB Memory
50 to 100	• 2 CPUs • 256 MB Memory	• 2 CPUs • 512 MB Memory	• 4 CPUs • 1 GB Memory
100 to 250	• 4 CPUs • 512 MB Memory	• 4 CPUs • 1 GB Memory	• 6 CPUs • 2 GB Memory
250 +	• 8 CPUs • 1 GB Memory	• 8 CPUs • 1 GB Memory	• 8 CPU • 2 GB Memory

Recommended physical memory for the cache is the cache size setting times 1KB plus an additional 25% as a safety margin. For example, if the cache size setting is 45000, physical memory needed just for the cache would be (45,000 * 1KB) + 11MB = 56MB. If excessive memory swapping is observed, either reduce the cache size setting in iw.cfg or install more memory.

Global Report Center Requirements

The TeamSite Global Report Center requires approximately 5 MB of physical memory. The OpenDeploy Global Report Center has the same requirement. Therefore, approximately 10 MB of physical memory is required for the Global Report Center.

Disk Space Requirements

All servers are configured to have sufficient disk capacity for 760 MB of TeamSite program files (/iw-home) and five to ten times the total amount required for the Website content files to consume (/iw-store). This amount of disk space is required in order to store TeamSite metadata and multiple versions of your Website files.

Inode requirements

TeamSite requires a large number of inodes for efficient performance. To estimate how many inodes your server will require, use the following formula:

```
# inodes = (# branches)(# average files in staging area
per branch)(# average historical versions/file)

(3 + 3(% of files having extended
attributes)/100))(safety-factor)
```

For example, if the TeamSite server has three branches, with 20,000 files in the staging area of each branch, (on average), ten versions of each file in its history list (on average), seven percent of files have extended attributes, and a 1.5x safety factor:

```
# inodes   = 3 * 20,000 * 10 * (3 + 3*.07)* 1.5

           = 600,000 * 4.8

           = 2.9M inodes
```

Global Report Center Requirements

The TeamSite Global Report Center (installation is optional) requires an additional 25 MB of disk space, plus 10-50 MB for data storage.

7.7.2. Scalability

Scalability is handled through increasing the capabilities of the server. If multiple servers are needed due to excessive delays in service then two installations of TeamSite would be needed. This would work well if there are multiple branches to split off onto the separate installations. The best example would be to split up the Intranet and Internet branches of the SFA Website.

7.7.3. Reliability

TeamSite is not a critical production server and does not require high availability. The content is archived and is available for recovery.

7.8. Portal

The Portal component requires the following components:

- IBM HTTP Server
- Java Development Kit (JDK) 1.1.8_09a Production Release to host the Viador Information Center
- SQLNet - Client / server middleware to connect to the database repository
- An Oracle 8i database instance with at least 90MB of disk space

The Portal must be installed in a directory named ***infospc*** immediately below the Web server's ***http*** document root directory. The install program will suggest an installation location based on the information entered about the Web server. Install the Viador Portal Suite at the document root of the Web server.

Information regarding the owner of the user-repository tables and related database connection needs to be registered with the Viador Information Center (VIC). In order to store the information regarding user-repository tables with the Viador Portal Suite, the SFA system administrator needs to run the **SetRepLogin** shell script that is installed within the **util** subdirectory, under the **infospc** directory, found inside the document root of your Web server.

The Viador Information Center server requires a connection to a database to be established in order to start up. The Viador Information Center attempts 5 retries at 30-second intervals to establish a connection to the server. If the database starts up more than 300 seconds after Viador Information Center attempts to start up, a timeout situation may occur.

SFA users will need to run a Java enabled browser in order to use Viador's E-Portal suite. The certified browsers for each platform are listed in the following table.

Table 32 – Client Compatibility

Client Platform	Client Browser
Windows NT 4.0	Netscape 4.51, 4.61, 4.72, 4.08
	Internet Explorer 4.01 (4.72.3110.8) **, 5.0
Windows 95	Netscape 4.51, 4.61, 4.72, 4.08
	Internet Explorer 4.01 (4.72.3110.8) **, 5.0
Windows 98	Netscape 4.51, 4.61, 4.72, 4.08
	Internet Explorer 4.01 (4.72.3110.8) **, 5.0

Client Platform	Client Browser
Windows 2000	Internet Explorer 5.0

7.8.1. Security

The security services provided by the portal framework include Authorization and Authentication.

Use authorization is organized by individuals, roles or groups, and is used to control the level of granular access an individual has to a portal object. The individual may be granted read-only access, or may be granted the ability to publish, modify or delete a portal object.

Authentication is the process of uniquely identifying a specific individual. The portal framework authentication services are also used to define and enforce corporate security policies, such as password aging.

7.9. Knowledge Management

7.9.1. DRE Engine Configuration

Once the Autonomy Server has been installed there are two new directories. These include:

- The Installation Directory
- A directory with the Autonomy Server name containing the HTML pages for the Autonomy Server. This directory is found under the Web root directory

As well as these two directories there are two new files in the Scripts Directory which contain the CGI Scripts. The *Server*.CFG file and the *KnowServer*.EXE (*Server* being the name chosen for the installation).

The Installation Directory contains the DRE used for querying and indexing. This constitutes the back end of the Autonomy Server. The other two directories (Scripts Directory and the Directory with the Autonomy Server name) contain all the files for the front end of the Autonomy Server.

DRE.ini naming conventions

The DRE.ini file has the option to be renamed but there are various rules that apply. The DRE.ini file can be called *WhateveryoulikeDRE.cfg* however:

- If the system finds DRE.INI in its directory, it will connect to that engine
- If not, then if the GUIAdmin's name is *MyServerDreAdmin.exe* then it will look for *MyServerDre.cfg* and connect to it.

The DRE name must be synonymous with the INI file, e.g. *WhateveryoulikeDRE.inc* implies *WhateveryoulikeDRE.cfg*, and the GUI is *WhateveryoulikeADMIN.exe.* Basically, the GUI takes off the "Admin" at the end of its name, and appends ".cfg" to find the right file.

Configuring The Dynamic Reasoning Engine

This section includes details of the configuration of the Dynamic Reasoning Engine (DRE) and the use of the utilities supplied with the DRE which are provided for maintaining and servicing the system.

The DRE uses a single configuration file DRE.ini. This file has a similar format to a standard Windows INI file, with a number of sections defined with square brackets and each section containing a number of key=value pairs. Please be aware that where any key=value parameter that calls a value from else where, the value setting must be treated as case sensitive.

The several sections in the configuration file each govern different aspects of the operation of the DRE and are arranged in the following format:

- [MySecuritySection]
- [License]
- [Server]
- [Schedule]
- [Default]
- [Fields]
- [Cache]
- [IndexSummary]
- [IndexCache]
- [dbname]

For details on the individual sections, please consult Appendix A.

AutoIndexer

This section describes how to configure the Automatic Indexing process. Automatic Indexing is a continually running process, which performs operations from a queue file or directory(ies). You will need to specify the name of the queue file(s) / directory(ies) in the autoindexer.cfg file (detailed in the following subsection).

A Queue File contains a list of filenames with a full or relative path. Every time a new name or list of new names is appended to the list, they automatically get processed. In the case of the directory(ies), the Automatic Indexing process looks at any new files that appear in the directory together with any files that have been replaced or updated, and automatically processes them. In addition autoindexer can handle a few additional data formats, such as HTML, Word documents, PDF files etc.

Once the DRE is running, the Autoindexer program may be started as a service allowing new data content to be added dynamically. Communication between Autoindexer and the DRE is fully automated and, once invoked, the process may be left running all the time.

To stop and restart the process, the <AppName>.dirstat file together with the <QueueFile>.pos in the autoindexer Directory is deleted. The <AppName>.dirstat file keeps a list of all the files that have been processed while the <QueueFile>.pos records the current position in the queue of files to be processed. To reprocess the last file dealt with, replace the contents of the <AppName>.dirstat file with the contents of the <AppName>.dirstat.bak file. Also replace the contents of the <QueueFile>.pos file with the contents of the <QueueFile>.pos.bak file. The <AppName>.dirstat.bak file keeps a copy of the <AppName>.dirstat file before the last file was processed while the <QueueFile>.pos.bak file keeps a copy of the <QueueFile>.pos file before the last file was processed.

Also located in the Installation Directory, is the .log file. This file records all the actions performed by the Autoindexer.

The AutoIndexer will automatically attempt to import and index any NEW files that appear in the directory(ies). You should be aware of any applications that might create temporary files in the directory. For example, MS Word create temporary files in the current directory. If this is done in the directory polled by AutoIndexer, they will get indexed.

The Automatic Indexing process uses a single configuration file autoindexer.cfg, which is located in the Autoindexer directory with the name identical to the Autoindexer Executable file. This file has a similar format to a standard windows INI file, with a number of sections defined with square brackets and each section containing a number of key=value pairs. The three sections; [Configuration], [Default], [myjob], in the configuration file each govern different aspects of the operation of the DRE and are arranged in the following format.

- [Configuration]
- [Default]
- [myjob]

In the above, [myjob] represents the section containing details on a particular job named myjob.

Please be aware that any key=value parameter that calls a value from elsewhere, the value setting MUST be treated as case sensitive.

The Autoindexer is used for indexing idx files, importing to an idx file and indexing, importing, indexing and deleting and just deleting. If you want to do more than one action, then you need to create multiple jobs within your configuration file. You can also perform these actions within more than one DRE at a time, see the DRE host settings in the [Default] part of the autoindexer.cfg.

For additional details on the individual sections, please see Appendix B.

Server Sizing and Scaling

The DRE can scale by creating multiple instances of the DRE. These additional DRE's can reside on the same box or on a different server. Since the DRE process is multithreaded the DRE process can take advantage the multiple processors. The DRE instance will have an IP address, a query port number and a index port number. All spidering processed will communicate with the DRE using the index port number and all query requests will be handled through the query port. Since the DRE can be accessed via a IP address this allows the DRE to be distributed onto different machines.

All Autonomy modules that need access to the DRE use this method of communicating with the DRE. Certain thresholds of a DRE instance is that is can hold up to 1.5 million documents before another instance of the DRE needs to be created. The ratio of server processors to DRE is based on several factors. Those factors are query response, number of queries, indexing load from the spiders. The Autonomy logs and reports will show when another processor needs to be added to the configuration. If the server box cannot be expanded another server box can be added to the configuration.

The HTTPFetch can also scale by creating multiple instances of the HTTPFetch. These additional HTTPFetches can reside on the same server or on a different server. The HTTPFetch is single threaded and can not take advantage of multiple processor machines. The rule of thumb for configuring at HTTPFetch is no more than 150 spider jobs per instance of the HTTPFetch. There is also a one to one ratio between the processor and the HTTPFetch instance.

The following data diagram is based on the configuration stated in section 2.41. Autonomy focuses on the number of queries / second and simultaneous queries and then determines the number of users supported based on the average queries a user submits.

7.10. Directory Server

None

7.11. File Storage

None

7.12. Database Server

None

8 Applications Design

8.1. Web Browser

Not Applicable

8.2. Firewall

Not Applicable

8.3. Load Balancing

Not Applicable

8.4. Web Server

Not Applicable

8.5. Application Server

8.5.1. Web Application Design Model

This section presents a short overview of an application design guideline for Web applications consisting of servlets, JSPs, and JavaBeans.

The general structure or design pattern of a typical SFA Web application is shown in the following figure.

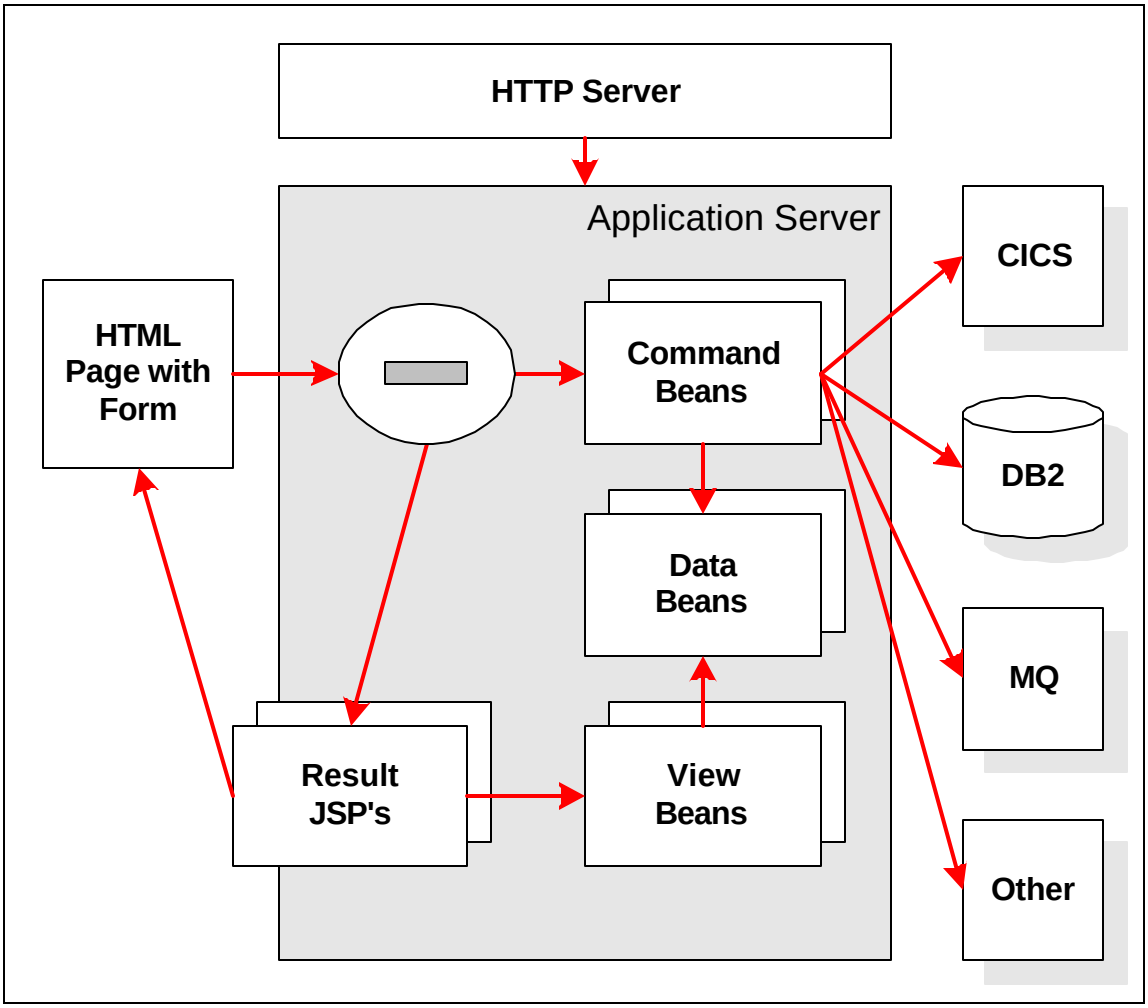

Figure 21 - Web Application Design Model

The major parts of such a design are discussed in the sequence of the flow of the application.

HTML page

The input page for each step is either a static HTML page or a dynamic HTML page created from a previous step. The HTML page contains one or multiple forms that invoke a servlet for processing of the next interaction. Input data can be validated through JavaScript in the HTML page or passed to the servlet for detailed validation.

Servlet

The servlet gets control from the Application Server to perform basic control of flow. The servlets validates all the data, and returns to the browser if data is incomplete or invalid. For valid data, processing continues. The servlet sets up and calls command beans that perform the business logic. The servlet initializes the view beans and registers them with the request block so that the JSPs can find the view beans. Depending on the results of the command beans, the servlets calls a JSP for output processing and formatting.

Command Beans

Command beans control the processing of the business logic. Business logic may be imbedded in the command bean, or the command bean delegates processing to back-end or enterprise systems, such as relational databases, transactions systems (CICS, MQSeries, IMS, and so forth).

A command bean may perform one specific function or it may contain many methods, each for a specific task.

Results of back-end processing are stored in data beans.

Data beans

Data beans hold the results of processing that was performed by the command bean or by back-end systems. For example, a data bean could contain an SQL result or the communication area of a CICS transaction.

Data beans may not provide the necessary methods for a JSP to access the data; that is where the view beans provide the function.

View beans

View beans provide the interface between the output producing JSPs and the data beans that contain the dynamic data to be displayed in the output. Each data bean contains one or multiple data beans and provides tailored methods so that the JSP has access to the data stored in the data beans.

JSPs

The JSPs generate the output for the browser. In many cases that output contains again form(s) to enable the user to continue an interaction with the application. JSP use tags to declare the view beans. Through the view beans the JSP gets access to all the dynamic data that must be displayed in the output.

Model - View – Controller

This design follows the model - view -controller design pattern:

- The JSPs (and HTML pages) provide the view,
- The servlet is the controller, and
- The command beans represent the model.

The data beans contain the data of the model and the view beans are helper classes to provide a data channel between the view and the model.

The servlet (controller) interacts with the model (the command beans) and the view (the JSPs). The servlet controls the application flow.

8.5.2. SFA Web Application Architecture

The sections that follow detail the recommendations in the following technology areas specific to the SFA Web Application Architecture:

- JavaScript
- JSPs and servlets
- Command beans
- JDBC / Structured Query Language for Java (SQLJ)
- Enterprise JavaBeans
- Connectors

JavaScript

JavaScript is a general-purpose object oriented programming language. It has great utility in Web applications because of the browser and document objects that the language supports. Client side JavaScript provides the capability to interact with HTML forms. You can use JavaScript to validate user input on the client and help improve the performance of your Web application by reducing the number of requests that flow over the network to the server.

To address various client side requirements, Netscape and Microsoft have extended their implementations of JavaScript in version 1.2 by adding new browser objects. Because Netscape's and Microsoft's extensions are different from each other, any script which uses JavaScript 1.2 extensions must detect the browser being used, and select the correct statements to run.

Versions 3.0 and earlier of both Netscape and Microsoft browsers don't support these new extensions. To run a script on most browsers, use JavaScript 1.1 that contains the core elements of the ECMAScript standard. Reference [1] is an excellent book on JavaScript that details the JavaScript objects and methods listing their origin and JavaScript level.

ECMA, a European standards body, has published a standard (ECMA-262) which is based on JavaScript (from Netscape) and JScript (from Microsoft) called ECMAScript. The ECMAScript standard defines a core set of objects for scripting in Web browsers. JavaScript is a superset of ECMAScript. It is comprised of the core ECMAScript objects that run on both Web browsers and servers, as well as a set of unique client specific and server specific objects. The core objects include array, date, math, number, and string. On the client side, there are document, form, frame, and window objects. These core objects enable the manipulation of HTML documents (checking form fields, submitting forms, and creating dynamic pages), and the manipulation of the browser (directing the browser to load other HTML pages, display messages, etc.).

JSPs and Servlets

The Framework supports the development of interaction controller and page construction logic using either Java servlets or Java Server Pages (JSP) technology. Both of these

implementation mechanisms have significant advantages over using CGI-BIN or Web server plugins. You will need to decide whether to use JSPs or servlets or both in your Web application.

JSPs can contain all the HTML tags that Web authors are familiar with. A JSP may contain fragments of Java code which encapsulate the logic that generates the content for the page. These code fragments may call out to JavaBeans to access reusable components and back-end data. To learn more about JSPs visit http://www.javasoft.com/products/jsp/.

Servlets are small Java programs that run on the Web Application Server. They interact with the servlet engine running on the Web Application Server through HTTP requests and responses. JSPs are compiled into servlets before being executed on the Application Server.

The interaction controller part of a Web application, which is primarily concerned with processing the HTTP request and invoking the correct business or UI logic, often lends itself to implementation as a servlet. This is especially true in cases where one interaction controller processes several types of user interactions. If your design dictates one interaction controller per type of user interaction JSPs are a better fit. For example, login would be handled by login.jsp, interaction1 by interaction1.jsp, interaction2 by interaction2.jsp and logoff by logoff.jsp.

JSPs were designed to simplify the process of creating pages by separating Web presentation from Web content. In the page construction logic of a Web application, the response sent to the client is often a combination of template data and dynamically generated data. In this situation, it is much easier to work with JSPs than to do everything with servlets.

Command Beans

Command beans provide a standard way to invoke a business logic request using a single round-trip message. A command bean is a JavaBean that encapsulates a single request to a target server (i.e., the server where the command is to be executed). The target server can be the same JVM as the client or a separate JVM.

Command beans allow for a clean separation of User Interface (UI) and business logic. The Web application parts (i.e. interaction controller and UI logic) can vary independently from the command bean's implementation.

The steps in a command bean's execution are:

- The JSP/servlet instantiates the command bean. It then sets the command bean's input properties. Finally, the JSP/servlet calls the perform method on the command bean
- The perform method, if successful, returns a copy of the command bean with its output properties set to the results of the underlying business logic task
- Control flows to the JSP/Servlet, which is now free to query the output properties of the command bean

JDBC / SQLJ

The business logic in a command bean will access information in a database for the database centric scenario. JDBC is a Java API for database-independent connectivity. It provides a straightforward way to map SQL types to Java types. With JDBC you can connect to your relational databases, and create and execute dynamic SQL statements in Java. JDBC drivers are RDBMS specific, provided by the DBMS vendor. Given common schemas between two databases an application can be switched between one and the other by changing the JDBC driver name and URL. A common practice is to place the JDBC driver name and URL information in a property or configuration file.

SQLJ provides a simplified syntax for JDBC that allows you to write SQL-like statements directly in your Java source code. The SQLJ preprocessor generates static SQL providing better performance than dynamic SQL. SQLJ will also generate iterator Java classes. These iterators allow you to navigate query results using a very simple "get next" protocol.

Enterprise JavaBeans (Release 2 Topology)

Enterprise JavaBeans (EJBs) are distinguished from JavaBeans in that they are non-visual, designed to be installed on a server, and accessed remotely by a client.

There are two types of Enterprise JavaBeans:

- Session
- Entity

A typical session Bean has the following characteristics:

- Executes on behalf of a single client
- Can be transactional
- Can update data in an underlying database
- Is relatively short lived
- Is destroyed when the EJB server fails. The client has to establish a new session Bean to continue computation.
- Does not represent persistent data that should be stored in a database
- Provides a scalable runtime environment to execute a large number of session Beans concurrently

A typical entity Bean has the following characteristics:

- Represents data in a database
- Can be transactional
- Shared access from multiple users
- Can be long-lived (lives as long as the data in the database)
- Survives EJB server crashes. A crash is transparent to the client

- Provides a scalable runtime environment for a large number of concurrently active entity objects

Typically an entity Bean is used for information that has to survive system crashes, while in session Beans, the data is transient and does not survive when the client's browser is closed. For example, a shopping cart contains information that may be discarded uses a session Bean, and an invoice issued after the purchase of the items is an entity Bean. An entity Bean very often makes direct calls on a database.

The business logic of a Web application often accesses data in a database. EJB EntityBeans are a convenient way to wrap the relational database layer in an object layer, hiding the complexity of database access. Because a single business task may involve accessing several tables in a database, modeling rows in those tables with EntityBeans makes it easier for command beans to manipulate the data.

Connectors

e-business connectors are gateway products that enable access to enterprise and legacy applications and data from your Web application. Connector products provide Java interfaces for accessing database, data communications, messaging and distributed filesystem services. The command bean model allows coding to the specific connector interface(s) while hiding the connector logic from the rest of the Web application.

8.6. Component Broker

TBD

8.7. Content Management

Choosing the right branch structure is a critical step in configuring a TeamSite installation. A good branch structure makes work processes efficient, because it minimizes interference between developers as they work on tasks concurrently, and eliminates extra steps involved in updating and synchronizing content from one branch to another.

8.7.1. Design Principles

Logically independent Website

In terms of content, we introduce the concept of a logically independent Website. By this we mean that the functionality of a logically independent Website is more or less complete, in terms of the files required to render that Website. For example, in a pure HTML Website, the HTML files and the graphics are contained within the logical Website. For a CGI-based Website, the CGI programs and scripts are contained within the logical Website. For an application server based Website the logical Website includes all the resources required by the application server, including the template files and the configuration files are contained in the logical Website. This means that the Website is separate from a content perspective.

Another property of a logically independent Website is that the developers working on that logical Website can release content changes more or less independently from other content outside the logical Website. This means that the Website is separate from a development perspective.

Lastly, there are few, if any links from outside a logical Website into that Website. The inbound links, if any, are to defined and stable pages. For example, a "press release" logical Website of a Website has few inbound links if other logical Websites of the Website point to the index page, or to a well-known URL from which all press releases can be rendered. On the other hand, it is less independent if other logical Websites point directly to press release pages. (The inbound links can be thought of the "API" of that logical Website of the Website.)

Task overlap

The second important concept is that of task overlap. By task overlap we want to assess how long it takes to start and complete a given change, with respect to other changes occurring to the Website at the same time. The key concept is to identify when a set of changes are released to the production Website, especially in regards to allow overlapping changes to go to production at or around the same time.

With overlapping tasks, determine whether the changes will be released at the same time or not. Changes for tasks with different release times should be kept separate to reduce interference. Changes interfere with one another when one change is complete and ready for approval or movement to production, but another change in the same workarea isn't complete, and hence prevents the first change from being approved or deployed to production.

There are different ways of keeping the changes separate. The simplest is to put changes for a task into separate workareas. There's a common pattern that keeps changes in a workarea. This can be done for example with workareas labeled Mon – Fri. Whatever content is staged for deployment in the respective workarea will be deployed at a scheduled time.

Basic Branch Patterns

This section introduces three common branch patterns, and a fourth special pattern that finds occasional use:

- Single-branch pattern
- Agency pattern (multiple independent branches)
- Long-term / short-term pattern
- Dependent-branch pattern

These patterns form the fundamental building blocks of more complex branch structures. Though they might seem simple, applying these patterns requires a solid understanding of the concepts introduced earlier.

These patterns form the fundamental building blocks of more complex branch structures. Though they might seem simple, applying these patterns requires a solid understanding of the concepts introduced earlier.

Single-branch pattern

The most common branch structure is the single-branch configuration. The earlier discussion regarding logically independent Website and task overlap comes into play because we arrange for a branch to hold a logically independent Website. In addition, we use the results of the task overlap analysis because the single branch needs to have a development flow that is consistent with having a single staging area. Specifically, this means that the task overlap is relatively small. For example, all tasks start and complete within a week or a few days. In contrast, a situation for which the single branch pattern wouldn't apply is one where half the Web team is working on a major revamp of the Website, while the other half pushes out incremental fixes and regularly scheduled content changes. This latter situation might be better suited for the long-term / short-term branch pattern discussed in the next section.

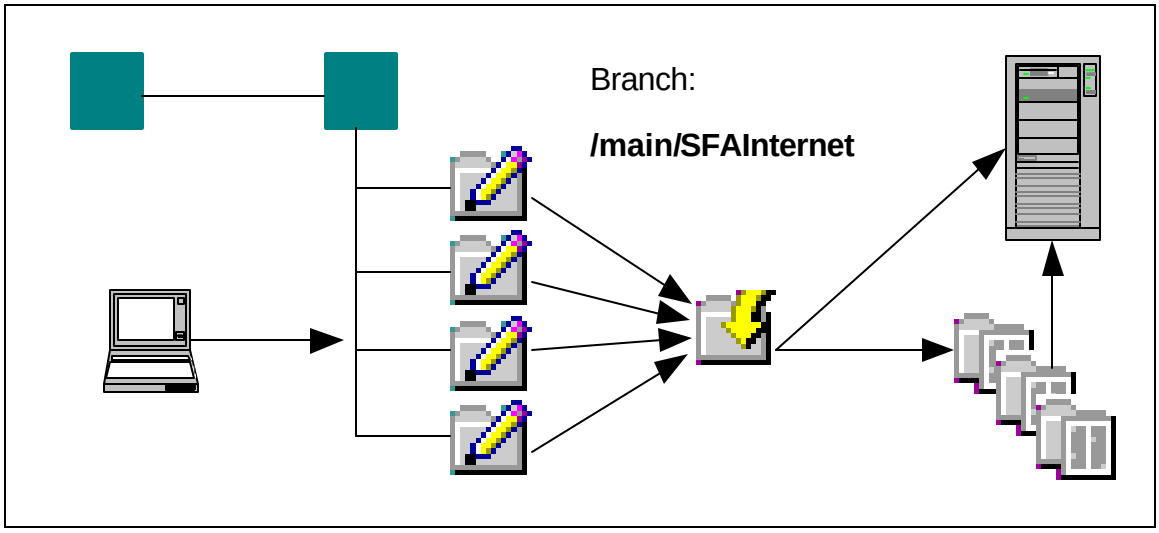

Figure 22 - Logically independent Web content goes into single-branch

Once we've determined that the logically independent Web content that goes into a branch, and we've analyzed the workflow to find the typical tasks and the overlap, we find a workarea structure that is compatible with that flow. Within the branch, there are a number of common ways to organize the workareas. The simplest is to do the work for a single task within a workarea. This might be a Web developer working independently, or this could be a small team consisting of an HTML developer and an artist.

Tasks should be distinguished by whether it interferes with the functionality of the Website, whether it interferes with the review and approval of the change, and when it will be submitted. If a task interferes with other tasks, changes for that task should be placed into its own workarea.

Workarea configurations

A workarea in a branch corresponds to a single task, where a task consists of a set of interrelated changes to Web content. Here are some examples of tasks:

- A single developer makes an HTML change
- A Web designer and a graphic designer collaborate on new pages
- A developer changes the logic in C++ files to fix a bug
- Several SFA managers create press releases, all of which are scheduled to go-live on the same day

Notice that in the first and third examples a single developer uses a workarea.

In the second and fourth examples several people work in the same workarea.

In the second example a Web designer and a graphic artist collaborate on new pages. In this case, it makes sense for them to work in the same workarea, because their changes don't interfere. For example, the Web designer will change the HTML, while the artist will change the images. In the fourth example the SFA managers make changes independently, but the commonality is that all of their changes go-live on the same day.

Per project workarea

A very common situation is when a single developer, or a small group working closely together, have changes that are interrelated, and which are submitted at the same time. They are interrelated because the assets reference one another, or they use one another. For example, when an HTML developer and a graphic artist develop assets, the HTML code might refer to an image. Similarly, Perl code might use a related Perl module. In this situation, it makes sense to use a separate workarea for each such task.

After the changes for a project have been submitted, the workarea can be reused for the next project. That's why it often makes sense to give each developer a workarea. In this case, a workarea is used for a succession of non-overlapping projects.

If a single developer has multiple ongoing projects, where the changes from one have different submission times, then it may make sense for that developer to have several workareas. For example, in the figure above, Richard has two workareas that she can use for changes that he needs to keep separate, because they have different due dates.

Per time slot workarea

Another common situation is when there is content that has a strong time component to it. For example, a press release goes live on a specific day, or a product promotion begins and ends on specific days. In this situation, it sometimes makes sense to define one workarea per day, with 31 workareas to correspond to each day of a month. For example, a workarea called, "day12" contains all content that is scheduled to go to production on the upcoming 12th of the month.

In the simplified usage of this technique shown below, there is a workarea per day of the week.

Agency pattern

The agency pattern is used when we have multiple logically independent Websites. This pattern gets its name because Web development agencies produce Websites for many clients. Since each client's Website is independent from each other, each Website becomes a project onto itself. To keep each project separate, each project uses a different TeamSite branch. Although this pattern is easy to spot in the case of Web development agencies, it is common to see this pattern in other situations as well.

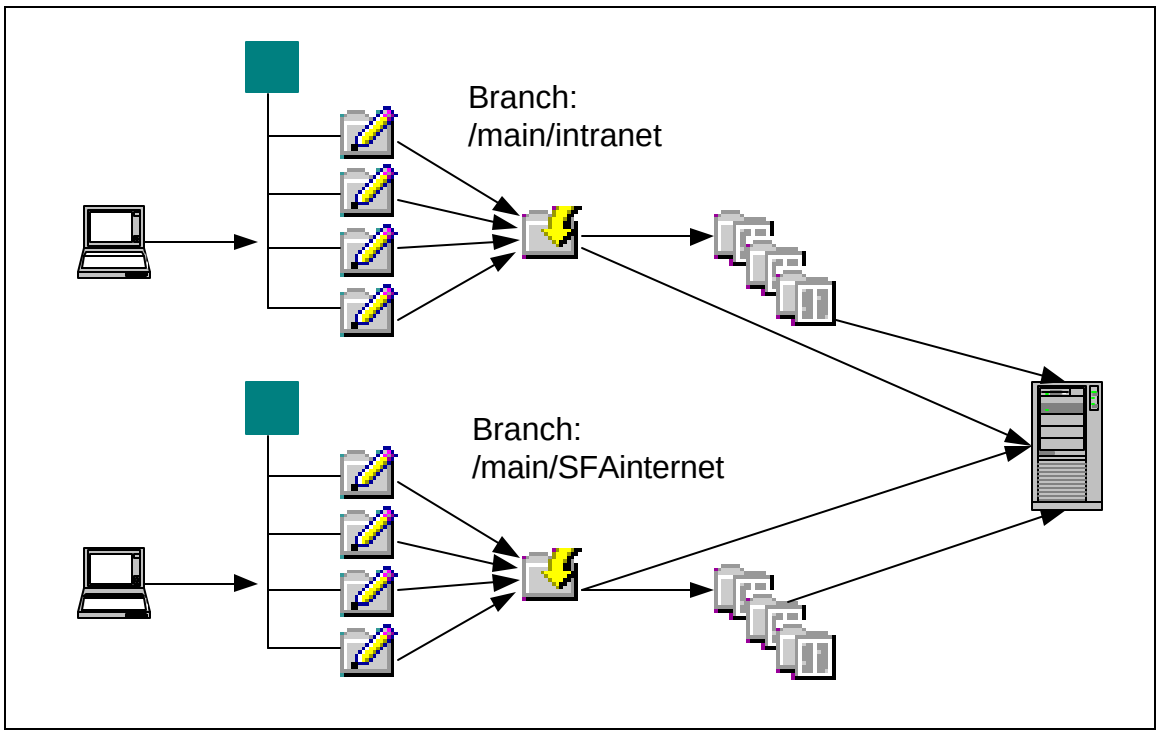

Figure 23 - Multiple independent Website, or "agency pattern.

For example, let's take the case of a Website for a consumer retail Website, which has a single point of entry through a corporate homepage. After detailed discussions with the Web development team, the Web content and work flow indicates that the Website can be decomposed into logically independent Websites for the corporate information section, including press releases and job listings, as distinct from the e-commerce section of the corporate Website.

In this hypothetical example, it is appropriate to decompose what is one large physical Website, into two logically independent Websites: one for the static section and another for the application server driven e-commerce Website. The information section and the e-commerce sections of the Website are independent from a content perspective, because each doesn't rely on the other from content, graphics, and application logic. Similarly, the two

sections are independent from a development perspective because they are done by different groups of developers, and the review and approval processes are distinct.

It is important to stress that identifying the logically independent Websites requires both studying the external look of the Website, and gaining an understanding of the internal organization of the content and the teams that develop the content.

If a logical Website satisfies the "independence" criteria in the content analysis, then it is a candidate to be version controlled in its own branch. In this case, developers are assigned to work in certain branches, and can be kept out of other branches. This is essence of the agency pattern.

Short-term/long-term branch pattern

The long-term / short-term pattern is used when there's a long-term Web development effort going on concurrently with short-term changes to a Website. The essential point is that there are changes for the long-term branch that overlap with changes for the short-term branch, and the changes cannot go to production together. For this reason, the development efforts are split apart and done separately.

Sometimes it isn't possible to keep changes in a workarea. This typically occurs for two reasons. First, this occurs when changes occur in gradual steps, and the incremental steps need to be version controlled. Second, this occurs when different people or teams do changes, and the separate changes need to be version controlled in stages. If changes for a given task cannot be kept in a workarea, then we ask whether the changes will be released at the same time. For example, suppose there is a press release and a bug fix to the search engine, and they are both going out tomorrow morning. Two separate teams are doing the changes, and we let each team make their changes in separate workareas. By doing this, each team can get their work reviewed and approved separately. And we don't care in which order they complete their tasks. Eventually the work is integrated, and the changes are moved to production. If the changes overlap, but are going to production at the same time, then a single branch is sufficient.

This leaves us with the situation that changes overlap, changes cannot be kept in only in a workarea, and the changes are going to production at different times. In this case, we introduce separate branches. Typically this corresponds to the long-term / short-term pattern.

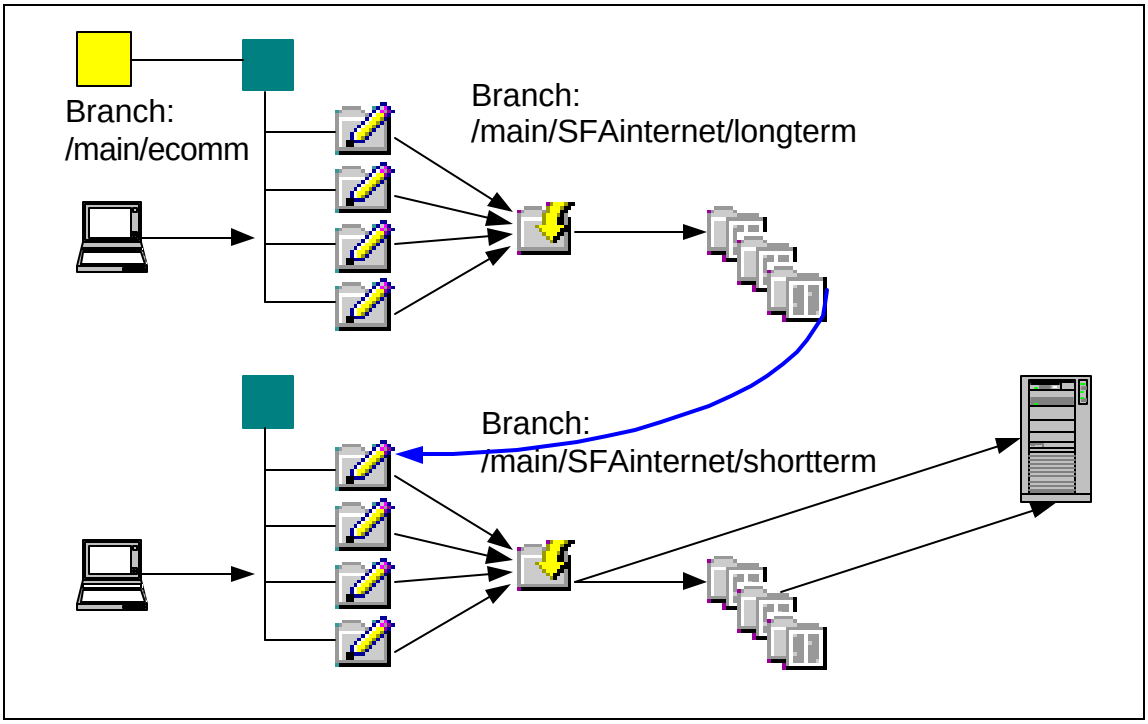

Figure 24 - The Longterm / Shortterm branch pattern

8.7.2. Dependent branch pattern

In the dependent branch pattern, a branch contains a set of Web assets that aren't logically independent. For example, the "/images" directory or the "/cgi-bin" directory are kept in a branch all to themselves. In particular, a branch that contains all the image assets, but nothing else, doesn't constitute a functioning Website. This pattern is used when there's a compelling reason to subdivide into branches what otherwise would be an independent Website:

- The volume of assets exceeds the amount that reasonably wants to be kept in a single branch
- There is a need to factor out assets for a common subsystem
- There is a strong organizational requirement to separate the assets
- There is a need to version control assets separately within a sub-branch

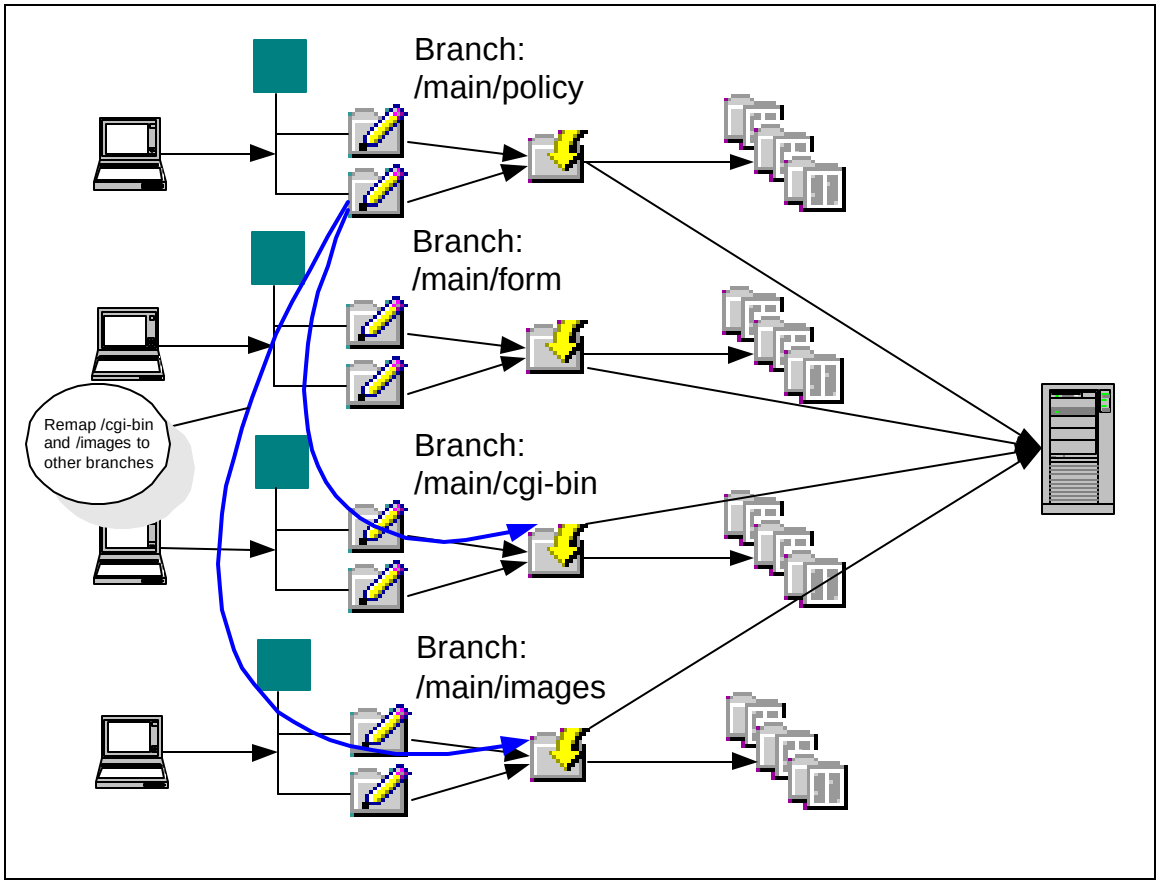

Figure 25 - The Dependent Branch Pattern

With a super large Website, the volume of assets that logically stands out as independent exceed what you'd want to manage as a single branch. For example, more assets in a branch mean that workarea update time is increased. To keep operation times within reasonable bounds, subdividing a super large Website into manageable dependent branches can be a viable solution.

Sometimes several logically independent Websites use a common subsystem, such as images or cgi-bin. In this situation, you'd prefer to not replicate copies of each of these subsystems into each branch. A good approach to this situation is to factor out the common subsystem as a dependent branch.

Occasionally organizations divide responsibilities of subparts of a Website very strongly, and want to reflect that in the organization of the assets. For example, a CGI development group is solely responsible for CGI development, and a separate content development group produces HTML assets. Ordinarily, the logically independent Website principle suggests keeping those subsystems together in a single branch, to facilitate the ability to test, quality assure, and approve the integrated Website. By separating these subsystems into branches, and if restrictions are placed on who can make changes in the branches, then an organization assures itself, for instance, that HTML changes originate solely from the HTML branch.

The need to perform additional versioning is another reason to form a dependent Website branch. This arises when a subsystem has content that needs to be versioned at checkpoints more finely grained than if they were in a single branch. For example, a file goes through an approval procedure with multiple reviewers. You want to checkpoint the file before the review starts. Changes resulting from the first review are checkpointed by submitting to the sub-branch, and similarly for the second review. At any stage of the process, versions of the file are available for comparison and rollback. When the reviews are completed, the file is propagated from the sub-branch to the upper branch.

Dependent Website branch arrangements typically introduces the need for additional proxy remap configuration, because without this some or all virtualization would be lost. For example, suppose the "/images" subsystem were factored apart from the "/htdocs" Web assets. What would otherwise be a functionally independent Website, would be no longer function alone. A reference to a file such as "/images/logo.gif" would be broken. To restore the link, a proxy rule that remaps a reference to "/images" in the htdocs branch should be introduced to map to the images branch.

Keep in mind that the proxy configuration will need to be synchronized with any directory and branch rearrangements. This introduces an additional level of maintenance effort. For example, if another dimension of Web asset factoring were introduced, say for the "/include" subsystem, then the proxy configuration will need to be revisited.

8.8. Portal

The Portal component provides the ability for tight integration of third party applications via a Java-based mechanism entitled the Viador Portlet Builder API. Portlets are content or application services that are registered with the Viador Information Center and can be controlled and deployed by any Viador portal interface. Content portlets include Web pages and content feeds (news and weather) while application services include calendars, email, and others. Portlet integration includes both simple launching from the portal by passing user parameters and using the Viador API based integration approach that will provide transparent application access.

The standards used for portlet coding is described in the following sections and will be reviewed as part of the review process when considering a Viador-compliant portlet.

8.8.1. Java Coding Conventions

All portlets will be coded according to the Java coding conventions as outlined from Sun Microsystems.

Enforcement of the coding conventions particularly in the area of commenting will be part of the review process of the portlet.

8.8.2. Standard Portlet Coding Template

Other Product Dependencies A standard code template will be used which includes all the standard header information about the portlet with built-in commenting sections that can be used. Portlets generally have the same skeleton of code that is used so a highly commented template can be used to give the Java coder the needed format to get started with coding. The code template will have the following items in the header section:

- Portlet Name
- Portlet Description
- Internet Dependencies
- User Administrator Settings
- Portlet Author(s)

A Java template header file, *portlet_template.java*, will be available as part of the Portlet Standardization Kit and it is strongly urged that this template be used as the starting point for portlet development.

A longer-term solution to having a coding template will be to develop an automated tool that can generate this template automatically with the correct naming conventions and cartridge information. This could be done as a Java program or even simpler would be a JavaScript interface.

8.8.3. Code Commenting

All portlet code must be commented extensively and in many cases over-commented so that the concepts can be easily looked at and understood by the novice programmer. This means it must go well beyond the comments that will be provided by the base template. The standard for commenting code is broken down into two areas, implementation comments and documentation comments.

Documentation comments can be fairly minimal but are valuable for the automated javadoc tools to automatically generate HTML pages. Documentation comments are denoted with the use of double asterisk comments, /**…*/ at the beginning of the comment. The Java template header file has documentation comments for the minimum requirements.

Implementation comments should be generous and produced roughly every 5-10 lines of to describe the code that is being executed. The comments themselves should be readable and clear. Standard implementation comments use the /*…*/ or // values. The Java template header file has default implementation comments in place and others ready to be filled in according to the portlet that is coded.

8.8.4. Portlet Generated HTML Pages and Error Messages

Many portlets will produce HTML pages as part of their defined actions. All HTML pages that are produced by the portlet will use standard HTML and will be very descriptive in their

output. HTML that is produced will be checked using the standard HTML validation programs available today.

The use of appropriate marketing approved logos and images in the HTML that is generated is encouraged to promote the branding of the portlet and its perspective interface.

All portlet error messages produced via HTML will be verbose and reviewed for clarity.

8.8.5. Debugging and Exception Handling

Portlets may contain debugging methods or code as long as they can be turned on or off.

All portlets should contain a general method for exception handling. A generalized exception handler would be desirable and can be made part of the standard coding template. A portlet could potentially effect the performance of the Viador server so proper exception handling is essential and if there are exceptions that could potentially show up on the Viador Server console they should identify themselves as a portlet based exception if possible.

8.8.6. Viador

The Viador Portlet Builder API is a Java-based mechanism that provides tight integration of 3rd party applications into the Portal. Portlets are content or application services that are registered with the Viador Information Center and can be controlled and deployed by any Viador portal interface. Content portlets include Web pages and content feeds (news and weather) while application services include calendars, email, and others. The Viador Portal Tools are also types of portlets, for example, User Administration, Document Courier and others

Portlet integration includes both simple launching from the portal by passing user parameters and using the Viador API based integration approach that will provide transparent application access.

8.8.7. Design Principles

The standard used for portlet coding is described in the following sections and will be reviewed as part of the review process when considering a Viador compliant portlet.

8.8.8. Java Coding Conventions

All portlets will be coded according to the Java coding conventions as outlined from Sun Microsystems.

Enforcement of the coding conventions particularly in the area of commenting will be part of the review process of the portlet.

8.8.9. Standard Portlet Coding Template

A standard code template will be used which includes all the standard header information about the portlet with built-in commenting sections that can be used. Portlets generally have the same skeleton of code that is used so a highly commented template can be used to give the Java coder the needed format to get started with coding. The code template will have the following items in the header section:

- Portlet Name
- Portlet Description
- Internet Dependencies
- Other Product Dependencies
- User Administrator Settings
- Portlet Author(s)

A Java template header file, *portlet_template.java*, will be available as part of the Portlet Standardization Kit and it is strongly urged that this template be used as the starting point for portlet development.

A longer term solution to having a coding template will be to develop an automated tool that can generate this template automatically with the correct naming conventions and cartridge information. This could be done as a Java program or even simpler would be a JavaScript interface.

8.8.10. Code Commenting

All portlet code must be commented extensively and in many cases over-commented so that the concepts can be easily looked at and understood by the novice programmer. This means it must go well beyond the comments that will be provided by the base template. The standard for commenting code is broken down into two areas, implementation comments and documentation comments.

Documentation comments can be fairly minimal but are valuable for the automated javadoc tools to automatically generate HTML pages. Documentation comments are denoted with the use of double asterisk comments, /**…*/ at the beginning of the comment. The Java template header file has documentation comments for the minimum requirements.

Implementation comments should be generous and produced roughly every 5-10 lines of to describe the code that is being executed. The comments themselves should be readable and clear. Standard implementation comments use the /*…*/ or // values. The Java template header file has default implementation comments in place and others ready to be filled in according to the portlet that is coded.

8.8.11. Portlet Generated HTML Pages and Error Messages

Many portlets will produce HTML pages as part of their defined actions. All HTML pages that are produced by the portlet will use standard HTML and will be very descriptive in their output. HTML that is produced will be checked using the standard HTML validation programs available today.

The use of appropriate marketing approved logos and images in the HTML that is generated is encouraged to promote the branding of the portlet and its perspective interface.

All portlet error messages produced via HTML will be verbose and reviewed for clarity.

8.8.12. Debugging and Exception Handling

Portlets may contain debugging methods or code as long as they can be turned on or off.

All portlets should contain a general method for exception handling. A generalized exception handler would be desirable and can be made part of the standard coding template. A portlet could potentially effect the performance of the Viador server so proper exception handling is essential and if there are exceptions that could potentially show up on the Viador Server console they should identify themselves as a portlet based exception if possible.

8.9. Knowledge Management

Not Applicable

8.10. Directory Server

Not Applicable

8.11. File Storage

Not Applicable

8.12. Database Server

Not Applicable

9 Additional Resources

The following sections provide resources for additional information.

9.1. Web Browser

Table 33 – Additional Resources: Web Browswer

Web Browser	Description
Microsoft Internet Explorer, Netscape Navigator, Lynx	
Microsoft® Pocket Guide to Microsoft Internet Explorer 5 /catalog/display.asp?title=4538&subid=22	The portable, reliable reference to Microsoft Internet Explorer 5, ideal for the frequent traveler or anyone seeking quick answers about the popular Web browser's tools, terms, and techniques.
Get Going with the Internet http://www.knowwareglobal.com/eng/internet_get_going.htm	Introduction to the Internet, its components and the browsers that can be used to achieve access to the Net.
Lynx Help for Beginners http://www.chass.utoronto.ca/%7Epurslow/lhfb.html	This outline is especially for people who are just starting to use Lynx or have used it for some time without exploring its features very far. It answers the sorts of questions everyone asks at first.

9.2. Firewall

Table 34 – Additional Resources - Firewall

Firewall	Description
Checkpoint	
Network Security Info Pack http://www.checkpoint.com/forms/newliter.htm	Information on Check Point's Network Security solutions. The package includes: FireWall-1 4.0 Brochure, VPN-1 Gateway Data Sheet, Stateful Inspection Tech Note, and the OPSEC (Open Platform for Security) Tech Note.
Redefining the VPN: **A White Paper** http://cgi.us.checkpoint.com/rl/resourcelib.asp?state=1&item=VPNWP	Document explaining typical VPN implementations and a helpful Buyer's Guide.
VPN Security Components http://cgi.us.checkpoint.com/rl/resourcelib.asp?state=1&item=VPNSec	Document describing the three essential components of a Virtual Private Network: security, traffic control, and enterprise management.

9.3. Load Balancing

Table 35 – Additional Resources – Load Balancing

Load Balancing	Description
IBM Performance Pack	
Review Guide http://www-4.ibm.com/software/Webservers/perfpack/library.html	This guide is designed to show you why IBM WebSphere Performance Pack for Multiplatforms V2.0 is the Web server environment of choice for Internet Service Providers and corporate technology specialists who need to reduce Web server congestion, increase availability, and improve Web server performance.
Getting Started http://www-4.ibm.com/software/Webservers/perfpack/library.html	Please note that this book contains links to other books that are shipped with Performance Pack. If you have not installed the other books, some links won't resolve. *User's guides* for other products can be found at their respective Websites.
IBM WebSphere Performance Pack: Web Content Management with IBM AFS Enterprise File System http://www-4.ibm.com/software/Webservers/perfpack/library.html	This Redbook will give you a clear understanding of the features of IBM AFS Enterprise File System, the File Sharing component of IBM WebSphere Performance Pack. It shows how to plan for, install, configure, use, tune and troubleshoot this component and offers specific implementation examples. Moreover, it helps explain how to build complex scenarios that involve all the components of IBM WebSphere Performance Pack, to give you a better understanding of the technologies involved.

9.4. Web Server

Table 36 – Additional Resources – Web Server

Web Server	Description
IBM HTTP Server	
Redbook Library http://www.redbooks.ibm.com/booklist.html	Contains all current IBM Redbooks. Not all books are available on hardcopy or CD-ROM.
Redpiece Library http://www.redbooks.ibm.com/redpieces.html	Contains IBM Redbooks under development and are published here for those who need the information now and may contain spelling, layout and grammatical errors.
Redpaper Library http://www.redbooks.ibm.com/redpapers.html	A developing resource of shorter technical documents.

9.5. Application Server

Installers and system administrators

If you need to install, configure, or manage a version of the WebSphere Application Server, read one or more of the following:

- To learn the basics of installing and configuring the Advanced Application Server, read *Getting Started with Advanced Editio*n. This document is designed for first-time users of the Advanced Application Server who want to get a simple system up and running quickly.
- To learn about managing the Advanced Application Server, access the Documentation Center and the online help available with the WebSphere Administrative Console.
- To learn the basics of installing CB, read *Getting Started with Component Broke*r; to learn the basics of installing and configuring TXSeries, read *Getting Started with TXSerie*s. These documents are designed for first-time users of Enterprise Application Server who want to get either CB or TXSeries up and running quickly.
- To learn about installing, configuring, and managing a more complicated system with the Enterprise Application Server, start with the *Planning, Performance, and Installation Guide* for CB or the *Planning and Installation Guide* for TXSeries.
- To learn how to use the adaptors available with CB, start by reading one or more of the following:
  - CICS and IMS Application Adaptor Quick Beginnings
  - Oracle Application Adaptor Quick Beginnings
  - MQSeries Application Adaptor Quick Beginnings

Application developers and system architects

If you need to design business systems or develop applications using a version of the WebSphere Application Server, read one or more of the following documents:

- To learn the basics of developing enterprise beans and related components in compliance with the Sun Microsystems Enterprise JavaBeans ™ Specification, start with *Writing Enterprise Beans in WebSphe*re. This document provides instructions for developing enterprise beans in both the Advanced Application Server and the Enterprise Application Server.
- To learn about the broader issues involved in designing and developing systems and applications in the WebSphere Family, read *Building Business Solutions with the WebSphere Famil*y.
- To learn about developing applications in CB, start by reading the *Application Development Tools Guide* and then read the Component Broker *Programming Guid*e.

Table 37 - Additional Resources – Application Server

Order Number	Book name	Book description
Common documentation for WebSphere Application Server Enterprise Edition (for all supported platforms)		
SC09-4430	*Introduction to WebSphere Application Server*	Provides a common familywide overview of all editions of WebSphere Application Server and the contents of each edition.
SC09-4431	*Writing Enterprise Beans in WebSphere*	Provides an introduction to programming with Enterprise JavaBeans in the Advanced and Enterprise Editions of WebSphere Application Server.

Order Number	Book name	Book description
SC09-4432	*Building Business Solutions with the WebSphere Family*	Provides programming examples and scenarios that demonstrate application development and recommended programming practices across the WebSphere Application Server family.

9.6. Component Broker

Table 38 – Additional Resources – Component Broker

Order Number	Book name	Book description
Common Component Broker documentation (for all supported platforms)		
SC09-4439	*CICS and IMS Application Adaptor Quick Beginnings*	Provides information on installing and verifying the application adaptor that provides communications between CB applications and CICS or IMS.
SC09-4440	*Oracle Application Adaptor Quick Beginnings*	Provides information on installing and verifying the application adaptor that provides communications between CB applications and an Oracle database.
SC09-4441	*MQSeries Application Adaptor Quick Beginnings*	Provides information on installing and verifying theapplication adaptor that provides communications between CB applications and MQSeries.
SC09-4442	*Programming Guide*	Provides information on developing CB applications in all supported programming languages on all supported platforms. It describes the programming model, including business objects, data objects, and includes information about various basic programming issues.
SC09-4443	*Advanced Programming Guide*	Describes CB's implementation of the Common Object Request Broker Architecture (CORBA)Object Service; the CB Object Request Broker (ORB); Cache, Notification, Query, Session, and other Services; interlanguage object model (IOM); and workload management.
SC09-4444	*MQSeries Application Adaptor Concepts and Development Guide*	Introduces the MQSeries Procedural Application Adaptor and provides information about developing applications that use both CB and MQSeries.
SC09-4445	*System Administration Guide*	Provides information on administering CB and CB applications on all supported platforms. Also provides general information about using the System Manager interface.
SC09-4445	*System Administration Guide*	Provides information on administering CB and CB applications on all supported platforms. Also provides general information about using the System Manager interface.
SC09-4446	*Programming Reference Volume 1*	With Volume 2, documents the complete CB application programming interfaces (APIs) available for all supported programming languages. In online formats (HTML and PDF), the *Programming Reference* is provided as a single volume.
SC09-4447	*Programming Reference Volume 2*	With Volume 1, documents the complete CB application programming interfaces (APIs) available for all supported programming languages. In online formats (HTML and PDF), the *Programming Reference* is provided as a single volume.

Order Number	Book name	Book description
SC09-4448	*Application Development Tools Guide*	Provides information about using the CBToolkit components, with a focus on common development scenarios such as inheritance and team development.
SC09-4449	*Problem Determination Guide*	Provides information to help administrators and programmers identify and solve problems with CB and CB applications. It includes information on installation problems, run-time errors, debugging of applications, and analysis of log messages.
SC09-4450	*Glossary*	Lists and defines terms commonly used in Component roker documentation.
GC09-4490	*Component Broker Release Notes*	Provides platform- and release-specific information about CB, including descriptions of new features that are more thorough than those in the CB README file, information for features or changes learned too late for incorporation into the product documentation, and information about known restrictions associated with CB and, where possible, suitable workarounds.
Component Broker for Solaris documentation		
SC09-4435	*Getting Started with Component Broker*	Provides simple, straightforward scenarios for setting up CB on Solaris.
SC09-4438	*Planning, Performance, and Installation Guide*	Provides complete instructions for installing and configuring the latest version of CB on Solaris.

9.7. Content Management

Table 39 – Additional Resources – Content Management

Content Management	Description
Interwoven	
http://www.beasys.com/press/releases/2000/0427_Interwoven_partner.html	Interwoven Partnership with BEA Systems
http://support.interwoven.com	Library of all Manuals for TeamSite
http://www.zdnet.com/products/stories/overview/0,8826,250441,00.html	
http://industry.java.sun.com/javanews/stories/story2/0,1072,21399,00.html	TeamSite classes offered through Intershop
http://www.intraware.com/ms/mktg/ds/iwov_inter_teamsite_ds.html - White Paper Datasheet on TeamSitefffa	Product Review on TeamSite, Pricing, Support etc.

9.8. Portal

Table 40 – Additional Resources – Portal

Portal	Description
Viador Portal	
Viador Product Datasheet http://www.viador.com/pdfs/Product_Data_Sheet.pdf	Summary of Viador's offerings, both technical and business value aspects.b
Viador Portal Suite Datasheet http://www.viador.com/pdfs/E-Portal_Suite.pdf	Summary of Viador's E-Portal Suite, technical details, business uses.
What is a Portal – Really? http://www.viador.com/portal/Viador_Portal_Presentation.ppt	This animated online presentation brings to life the e-portal, and illustrates its strengths, uses and business value.

9.9. Knowledge Management

Table 41 – Additional Resources –Knowledge Management

Knowledge Management	Description
Autonomy	
Doculabs' Executive Brief: Doculabs' Functional Review of Autonomy http://www.autonomy.com/tech/doculabsanalysis.htm	This Executive Brief provides background information on Autonomy, Inc., and presents Doculabs' analysis of Autonomy's KM technologies: Knowledge Server, Knowledge Update, and Knowledge Builder.
Buttler Group Report http://www.autonomy.com/tech/wp.html	Summary of Autonomy's Knowledge Suite, pluses and minuses, and technical details.
IDC's White Paper on Autonomy: Automating Content Integration with Autonomy http://www.autonomy.com/tech/idc.htm	This White Paper analyzes the role of unstructured content in E-Business applications, and how the Autonomy Content Infrastructure can serve to integrate this content across an existing E-Business architecture.
Autonomy Technology Whitepaper http://www.autonomy.com/tech/wp.html	Autonomy's technology offers a breakthrough in managing unstructured digital information, including word processing and HTML-based files, email messages, and electronic news feeds.

9.10. Directory Server

Table 42 – Additional Resources – Directory Server

Directory Server	Description
Netscape Directory Server	
Netscape Directory Server Administrator's Guide /docs/manuals/index.html?content=directory/41/ag/contents.htm	This guide explains how to administer the Netscape Directory Server 4.1. After configuring your server, use this manual to help maintain your server

Directory Server	Description
Netscape Directory Server Installation Guide http://developer.netscape.com/docs/manuals/directory/install30/contents.htm	This guide explains how to install the Netscape Directory Server 4.1. After configuring your server, use this manual to help maintain your server.
Netscape Directory Server Deployment Guide /docs/manuals/index.html?content=directory/41/de/contents.htm	Guide explains aspects of planning for and deployment of the Netscape Directory Server 4.1

9.11. File Storage

Table 43 – Additional Resources – File Storage

File Storage	Description
AFS	
Managing AFS – the Andrew File System http://www.amazon.com/exec/obidos/ASIN/0138027293/bookmag/102-5242491-4840951	Written for UNIX system administrators, this guide deals with AFS, a high-end UNIX filing system developed at Carnegie Mellon University and used widely in many industries such as banking and finance. This title explains how to manage AFS to its greatest effect including the installation of an adequate server setup to handle thousands of clients with a minimum of administrator and hardware overhead. .
IBM AFS Administration Guide http://www.transarc.com/Library/documentation/afs/3.6/unix/en_US/HTML/AdminGd/auagd002.htm	This guide describes the concepts and procedures that an AFS(R) system administrator needs to know. It assumes familiarity with UNIX(R) administration, but no previous knowledge of AFS.
IBM AFS Quick Beginnings http://www.transarc.com/Library/documentation/afs/3.6/unix/en_US/HTML/QkBegin/auqbg002.htm	This guide explains how to install and configure AFS(R) server and client machines. It assumes that the reader is familiar with UNIX(R) system administration, but not AFS.

9.12. Database Server

Table 44 – Additional Resources – Database Server

Resource	Description
Database Server - Oracle 8i	
Armaghan's SQL PlusPlus http://www.oracleprofessionalnewsletter.com	SQL PlusPlus adds more then 120 command to your Oracle SQL*Plus. It also contains a very powerful PLSQL Code Generator for generating production-ready PLSQL Code for Table APIs and Web.
OraDev.com - for Oracle Developers http://www.xs4all.nl/~defcom/index.html	Site with categorized links to the Oracle world, book description, scripts, hints and tips.
Adelante - Oracle DBA's and Consultants http://www.adelante.u-net.com/	Site contains Oracle scripts and tips, and an Oracle DBA Quiz.

Resource	Description
Case method and OCP revision notes http://www.pitchmark.demon.co.uk/case.htm	Contains Word 97 document templates for the case method and revision notes for the Oracle Certified Professional program.

10 Product Overviews

10.1. Web Browser

The Web Browser component is the user interface to the content provided by the SFA application. The Web Browser is any commercial COTS product capable of rendering HTML and that utilizes HTTP for communication to the SFA application. The typical component is a personal computing device (PC, etc.) supporting either Netscape Navigator or Microsoft Internet Explorer. The Web Browser component is expected to support SSL and may support some level of Dynamic Hypertext Markup Language (DHTML), except that DHTML is not being produced by the SFA application. The SFA application does not produce any mobile or active HTML components.

The Web Browser component is provided by each individual user and is configured appropriately for the user organization. Access to the IA applications is through a URL.

10.2. Firewall

The IA assumes the existence and operation of Firewall components to screen public user access and protect SFA private assets. The Firewall components are configured to produce an isolation network "demilitarized zone" (DMZ).

Firewalls provide services that can be used to control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- The Protocol Firewall
- The Domain Firewall

The Protocol Firewall and Domain Firewall nodes provide increasing levels of protection at the expense of increasing computing resource requirements. The Protocol Firewall is typically implemented as an internet protocol (IP) Router, while the Domain Firewall as a dedicated server node. The IA places certain Load Balancing components within the DMZ to deliberately restrict the proliferation of firewall port penetrations.

The Firewall component is provided by the VDC and its implementation is independent of the IA components. The following diagram illustrates the port penetrations required of the firewall components.

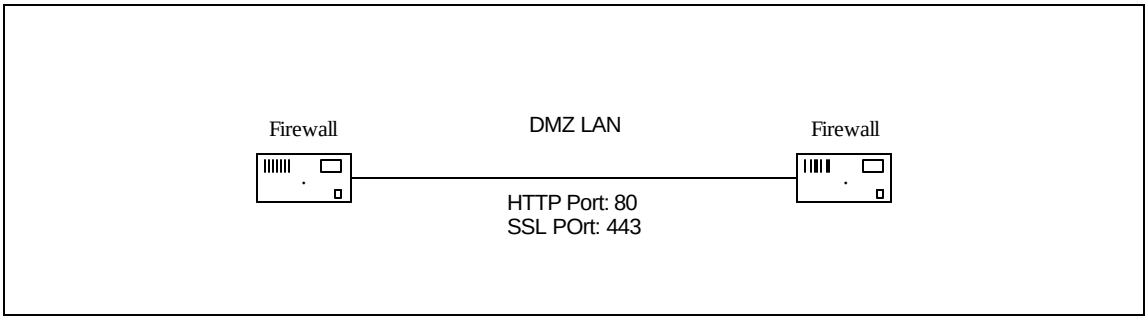

Figure 26 - IA Firewall Port Penetrations

10.3. Load Balancing

The Load Balancing component distributes workload across replicated components of the IA. The principal workload distribution opportunities are URL routing and portal brokering.

The IBM SecureWay ND COTS product is utilized for intelligent routing of HTTP URL requests across the IHS set. ND performs load balancing using a sophisticated workload assessment mechanism. The product is installed within the Intranet and directs IP packets from the virtual Website IP address to the IHS Website cluster. ND is aware of IHS status and redirects IP traffic to avoid a failed IHS.

ND is configured in a high availability master and slave redundant configuration. In this configuration the master ND performs the IP redirection and the slave replica monitors the master. When the master fails, the slaves negotiate and a slave asserts as master.

The Load Balancing component is implemented by the IBM WebSphere Performance Pack (Performance Pack) and provides a facility to reduce Web server congestion, increase content availability, and improve Web server performance. Performance Pack is used across the IBM WebSphere Application Server editions. It provides sophisticated detection of system utilization and error events across multiple networks and servers. It is extremely robust and scalable, providing caching of content across multiple servers and automating the replication and mirroring of data and applications. Performance Pack is geared for Internet Service Providers (ISP), whether they are in the ISP business or in the business of providing access to internal users of corporate information technology (IT).

ND improves the performance of servers by basing its load balancing decision not only on the servers' availability, capability and workload, but also on many other user-defined criteria as well. Using ND, the SFA Websites can take advantage of differentiated qualities of service, based on request origin, request content and overall load on the system. The entire load balancing operation is transparent to end users and other applications.

The ND component of IBM WebSphere Performance Pack consists of three sub-components that can be used separately or together to provide superior load-balancing results:

- Dispatcher
- Interactive Session Support (ISS)

- Content Based Routing (CBR)

The Dispatcher component can be used by itself to balance the load on servers within a local area network (LAN) or wide area network (WAN) using a number of weights and measurements that are dynamically set by Dispatcher. This function provides load balancing at a level of specific services, such as HTTP, file transfer protocol (FTP), secure socket layer (SSL), network news transport protocol (NNTP), post office protocol 3 (POP3), simple mail transfer protocol (SMTP), and Telnet. It does not use a domain name server (DNS) to map domain names to IP addresses.

The ISS component can be used by itself to balance the load on servers within a local or wide area network using a DNS round-robin approach or a more advanced user-specified approach. Load balancing is performed at the machine level. ISS can also be used to provide server load information to a Dispatcher machine.

When used for load balancing, ISS works in conjunction with the DNS server to map DNS names of ISS services to IP addresses. When used to provide server load information, a DNS is not required.

The CBR component works along with Web Traffic Express (WTE) to load balance client Web requests to specified servers; the routing is determined by comparing the content of the request to rules that have been defined in the CBR component.

10.3.1. How the Dispatcher Function Works

The Dispatcher creates the illusion of having just one server by grouping systems together into a cluster that behaves as a single, virtual server. The service provided is no longer tied to a specific server system. Therefore, systems can be added and removed from the cluster, or can be shut down for maintenance or other purposes, while maintaining continuous service. For the service requesters (clients), the balanced traffic among servers seems to be a single, virtual server, and the site appears as a single IP address to the world. All requests are sent to the IP address of the Dispatcher server, which decides for each client request which transmission control protocol (TCP) server is the best one to accept requests, according to certain dynamically set weights. The Dispatcher routes the client's request to the selected TCP server, and then the server responds directly to the client without any further involvement of the Dispatcher. The Dispatcher can also detect a failed server and route traffic around it.

The Dispatcher receives the packets sent to the cluster. These packets have a source and a destination address; the destination address is the IP address of the cluster. All servers in the cluster and in the Dispatcher system have their own IP address. In addition, they have an alias for the IP address of the cluster. The Dispatcher system has the cluster address aliased on the network interface, while all the TCP servers that will be load balanced by this ND machine have the cluster address aliased on the loopback adapter. The Dispatcher system check which server is the next best server to handle the load and routes the packet to that server. The Dispatcher routes this request based on the hardware address of the network adapter (Media Access Control (MAC) address) of the chosen server. It changes the hardware

address of the packet to the hardware address of the selected server and sends the packet to the server. However, the Dispatcher does not change the source and destination IP addresses in the packet. The server receives the packet and accepts it because all servers in the cluster have an alias for the cluster's IP address on the loopback interface. Then, the server sends a response back to the client by inverting the source and destination IP addresses from the original packet received. This way, the server can respond directly to the client.

The fact that the server can respond directly to the client makes it possible to have a small bandwidth network for incoming traffic, such as Ethernet or token-ring, and a large bandwidth network for outgoing traffic, such as Asynchronous Transfer Mode (ATM) or Fiber Distributed Data Interface (FDDI).

10.3.2. ND Component

The three components that make up the Network Dispatcher (ND) are the Dispatcher, ISS and CBR. ND provides the flexibility to use these components separately or together.

Dispatcher Overview

The Dispatcher component does not use a DNS for load balancing. It balances traffic among the servers through a unique combination of load balancing and management software. The Dispatcher can also detect a failed server and forward traffic around it.

All client requests sent to the Dispatcher server are directed to the TCP server selected by the Dispatcher as optimal according to certain dynamically set weights. The values for those weights can either be left as default for changed during the configuration process. The Dispatcher has two important features:

- The TCP server sends a response back to the client without any involvement of the Dispatcher
- This configuration does not require any changes on the TCP servers to communicate with the Dispatcher

With the Dispatcher, many individual servers can be linked into what appears to be a single, virtual server. The site thus appears as a single IP address to the world. Dispatcher functions independently from a DNS in that all requests are sent to the IP address of the Dispatcher server.

Dispatcher High Availability

The Dispatcher offers a built-in high availability feature. Dispatcher high availability involves the use of a second Dispatcher machine that monitors the main, or primary, machine and stands by to take over the task of load balancing should the primary machine fail at any time.

Interactive Session Support (ISS) Overview

The ISS can be used with or without a DNS name server. If ISS is used for load balancing, a DNS server is required. This can either be an actual DNS server or, a separate sub-domain for a new name server, a replacement name server provided by ISS. Using this approach, ISS runs on a DNS machine. A client submits a request for resolution of the DNS name for an ISS-associated service, which has been set up by an administrator. ISS then resolves the name to the IP address of a server in the cell, and forwards this IP address to the client. ISS does not require a DNS server to collect server load information.

The ISS monitor collects server load information from the ISS agents running on the individual servers and forwards it to the Dispatcher. The Dispatcher uses this load information, along with other sources of information, to perform load balancing.

ISS periodically monitors the level of activity on a group of servers and detects which server is the least heavily loaded. It can also detect a failed server and forward traffic around it.

Once every monitoring period, ISS ensures that the information used by the DNS server or the Dispatcher accurately reflects the load on the servers. The load is a measure of how hard each server is working. The system administrator controls both the type of measurement used to measure the load and the length of the load-monitoring period, taking into account such factors as frequency of access, the total number of users, and types of access (e.g. short queries, long-running queries, or central processing unit (CPU) intensive loads).

ISS High Availability

ISS supports high availability. This is accomplished by having all the nodes working together to eliminate any single point of failure. In case of a machine failure, a new monitor is automatically elected to take over.

Content-Based Routing Overview

CBR relies on WTE, a caching proxy server, to be installed on the same machine as CBR. WTE allows the manipulation of the caching details for faster Web document retrieval with low network bandwidth requirements. CBR, along with WTE, filters Web page content using specified rule types.

CBR provides the ability to specify a set of servers that should handle a request based on regular expression matching the content of the request. CBR also allows for multiple servers to be specified for each type of request. The requests can be load balanced for optimal client response. CBR can detect when one server in a set has failed, and stop routing requests to that server. The load-balancing algorithm used by the CBR component is identical to the proven algorithm used by the Dispatcher component.

When the WTE proxy receives a request, it is checked against the rules that have been defined in the CBR component. If a match is found, then one of the servers associated with that rule is chosen to handle the request. WTE then performs its normal processing to proxy the request to the chosen server. CBR has the same functions as the Dispatcher with the

exception of high availability, subagent, wide area network support, and some configuration commands. CBR can only function as part of WTE. WTE must be running before any CBR configuration can be performed.

The initial release of the IA does not utilize CBR and does not require WTE.

10.3.3. Dispatcher Functional Components

The Dispatcher consists of three main functions:

Executor - This function supports port-based routing of TCP and UDP connections to servers based on the type of request received (e.g. HTTP, FTP or SSL). This module always runs when the Dispatcher function is being used.

Manager - This function sets weights used by the Executor based on internal counters in the Executor itself and feedback from the Advisors and ISS monitoring (if ISS is used as a monitoring tool). Each unit of information given to the Manager by the Advisors, ISS and Executor has a relative importance, so more importance to one unit of information can be given over the others, or totally ignore one or more units of information. Using the Manager is optional, but if the Manager is not used, load balancing is performed using weighted round robin scheduling based on the current server weights.

Advisors - Advisors send requests to the TCP servers to measure actual client response time for a particular protocol. These results are then fed to the Manager to adjust the load balancing weights. Currently, there are Advisors available for HTTP, FTP, SSL, SMTP, NNTP, POP3 and Telnet. Three new Advisors have also been added: ping, WTE, and workload management (WLM).

10.4. Web Server

The Web Server component accepts HTTP URL connection requests from the Web Browser components. The Load Balancing component directs URL requests to a Web Server based upon workload and Web Server availability.

The Web Server determines whether the URL denotes content that is static or dynamic. Static content may already be available to the Web Server within a cache and is immediately returned. Dynamic and non-cached static content is transferred to an Application Server associated with the appropriate application domain. Static content is then retrieved and returned and dynamic content is assembled and returned.

The Web Server and Application Server URL transfer is managed by the Servlet Redirector. The Servlet Redirector manages the available Application Servers within the appropriate domain and preserves sessions. The Web Server component communicates to the Application Server component using the Remote Method Invocation (RMI) protocol. The transport mechanism is the Internet Inter-Orb Protocol (IIOP).

The Servlet Redirector may be installed in a Thin or Thick configuration. The Thin Servlet Redirector configuration provides the most efficient solution but does not allow for dynamic administration. The Thick Servlet Redirector configuration provides the least efficient solution but does permit for dynamic administration. The SFA solution requires the Thick Servlet Redirector configuration.

10.5. Application Server

The Application Server component is a functional extension of the Web Server component and is implemented with IBM WebSphere Application Server (WAS). It provides the technology platform and contains the components to support access to both public and user specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way it acts as an interface to business functions.

This component is within the enterprise network for security reasons. In most cases, access to this server would be in secure mode, using services such as SSL or Secure IP (IPSec).

In the simplest design, this component manages hypermedia documents and diverse application functions. For more complex applications or those demanding stronger security it is recommended that the application be deployed on a separate application server node.

Data that may be contained on the node include:

- HTML text pages, images, and multimedia content to be downloaded to the client browser
- Java Server Pages

IBM WAS extends the functionality of the IBM HTTP Server. WAS enables Web transactions and interactions with a robust deployment environment for SFA applications. It provides a portable Java-based Web application deployment platform focused on supporting and executing servlets, JavaBeans, and JSP files.

In particular, WAS provides:

- Support for JavaServer Pages, including:
  - Support for specifications 0.91 and 1.0
  - Extended tagging support for queries and connection management
  - An Extensible Markup Language (XML)-compliant document type definition (DTD) for JSPs
- Support for the Java Servlet API 2.1 specification including automatic user session and user state management
- High speed pooled database access using Java Database Connectivity (JDBC) for DB2 Universal Database and Oracle 8i
- XML server tools, including a parser and data transformation tools

- A Website analysis tool for developing traffic measurements to help improve the performance and effectiveness of your Websites
- Machine translation for dynamic language translation of Web page content
- Tivol Ready modules
- Additional integration with IBM VisualAge for Java to help reduce development time by allowing developers to remotely test and debug Web-based applications
- Full support for the Enterprise JavaBeans (EJB) 1.0 specification
- Deployment support for EJBs, Java servlets and JSPs with performance and scale improvements , including:
  - Applet-level partitioning
  - Load balancing
  - Enhanced support for distributed transactions and transaction processing
  - Improved management and security controls, including:
    - User and group level setup
    - Method level policy and control
- Common Object Request Broker Architecture (CORBA) support, providing both bean-managed and container-managed persistence
- Full distributed object and business process integration capabilities
- IBM's world-class transactional application environment integration (from TXSeries)
- Full support for the EJB 1.0 specification
- Complete object distribution and persistence (from CB)
- Support for MQSeries
- Complete component backup and restore support
- XML-based team development functions
- Integrated Encina application development kit

IBM WebSphere Application Server Enterprise Edition Product Overview

IBM WebSphere Application Server (WAS) Enterprise Edition (EE) provides the following basic capabilities and services:

- Supports enterprise-capable e-business initiatives from Web self-service to business integration to e-commerce
- Meets strict enterprise requirements for security, performance, scalability and configurability with open, standards-based technology like interoperable CORBA and EJB.
- Offers transactional integrity across multiple databases and other existing back-end transaction environments
- Enables integration of existing IT applications and resources through operational reuse with new business applications composed from existing ones

- Supports distributed object computing through a scalable, manageable runtime environment for developing and deploying distributed, component-based applications

WAS EE provides server software capable of handling high-volume, Web-based transaction processing. It enables migration of SFA business processes to the Web and positions the SFA IT infrastructure for future application growth.

WAS EE enables Web interactions while integrating with legacy SFA enterprise systems. Offering the highest levels of security, performance, scalability and availability, WAS EE supports distributed applications with efficient mainframe interoperability. IT supports development, deployment and management of enterprise-wide high performance business applications.

WAS EE supports and executes Java servlets, JavaBeans, JSP and EJB while interacting with the enterprise databases, Legacy processing systems and other SFA applications in producing dynamic Web content.

WAS EE builds Websites capable of handling advanced, sophisticated applications with features that include:

- Complete Java support including a server for applications built to the EJB specification for server-side component applications.
- An IBM HTTP Server, based on the Apache Web server, plus support for the other major HTTP Web servers.
- Performance and scaling attributes with support for bean-managed and container-managed persistence, providing relational database transaction management and monitoring. Container management and persistent storage (with the DB2 Universal Database) help provide a high-performance transactional environment using servlets, EJB, CORBA C++ and Java distributed objects.
- Tivoli -ready modules that can be managed by Tivoli-based tools.
- Site analysis tools to provide valuable information about SFA customers and the effectiveness of the Web-enabled SFA applications. Analysis results can be used to help improve site content, manage site integrity, analyze visitor behavior and better understand Website usage. With a rich set of analysis features and customizable options, the site analysis tools help maximize the return on the SFA Website investment.
- Automated machine translation features included IBM HTTP Server, providing “on the fly” language translation. Machine-based translation automatically translates the textual content of Web pages without human intervention, reducing the time and expense associated with human-based translation efforts.

WAS EE contains all of the products found in the Advanced Edition and adds the following major product components:

- CB is an enterprise solution for distributed computing, providing a scalable, manageable run time for developing and deploying distributed component-based solutions. It is a complete and integrated implementation of the open standards contained in the Object Management Group’s (OMG) CORBA. In addition, CB contains a separate

implementation of the EJB Specification, which can be used with or instead of the implementation contained in the Advanced Edition.

- TXSeries, which contains two middleware packages, Customer Information Control System (CICS) and Encina, that support and simplify the creation of transactional applications that can span multiple platforms in diverse and complex networks. In addition to offering cross-enterprise integration, TXSeries applications provide high levels of scalability, availability, integrity, longevity, and security. WAS EE contains a complete tool set for building applications.

The next two sections examine the components of WAS EE and explains what each of these components does. The remainder of this section explains some of the major underlying environments and services on which WAS EE runs. It also briefly discusses some of the other products that are licensed for use with WAS EE.

The WAS EE relies on one or more of the following services to handle low-level tasks such as security, naming, and remote procedure calls:

- The OMG Distributed Computing Environment (DCE).
- The OMG CORBA.
- Microsoft Corporation's Component Object Model (COM).
- WAS also contains an implementation of the EJB Specification (and related Java specifications) that is built into the CB product.

DCE enables distributed transaction processing environments using CB or TXSeries to run seamlessly across machines that differ in hardware, operating system, network transport, and application software. It is utilized internally as a system resource and is not visible in scope beyond IBM CB. The DCE deployment is specific to IBM CB and does not require installation configuration.

The DCE layer extends the basic operating systems of the separate machines to provide a common infrastructure for distributed computing. By using the standard interfaces provided by DCE, applications can operate within and be ported to other DCE platforms. The following sections describe the DCE services used by WAS EE.

Remote Procedure Call (RPC)

At the core of DCE support is the remote procedure call (RPC). RPCs provide a form of network-transparent communication between processes in a distributed system. Processes use RPCs to communicate in exactly the same way regardless of whether they are on the same machine or different machines in an administrative unit known as a cell. The DCE Security Service can be used to authenticate RPCs. Authenticated RPCs can be checked for tampering and can be encrypted for privacy. DCE uses multithreading to enable a client to have multiple concurrent RPC conversations with servers and to enable a server to handle many concurrent client requests.

Cell Directory Service (CDS)

The cell directory service (CDS) provides a namespace within which network resources can be accessed by name only. Applications do not need to know the addresses of resources. (Typical network resources are servers, users, files, disks, or print queues.) Further, if a resource is moved, it can still be located by the same name; application code does not need to be changed. The CDS Server manages a database, called a clearinghouse, which contains the names and attributes (including locations) of network resources in the DCE cell. When a request is made for a network resource, the CDS Server dynamically locates the resource. The DCE Directory Service also supports a global name service for identifying resources outside a cell.

DCE Security Service

The DCE Security Service provides secure communications and controlled access to network resources in a DCE cell. It verifies the identity of DCE principals (users, servers, and DCE-enabled clients) and allows them to access only the network resources that they are authorized to use. The DCE security service does the following:

- Manages a central source of security information in the cell's security database.
- Validates the identity of interactive principals, such as users, by enabling them to log into DCE. This is known as establishing a login context.
- Grants *tickets* to principals and services so their communications are secure.
- Certifies the credentials of principals to control principals' access rights to resources.
- Validates the identity of non-interactive principals, such as CICS regions, by enabling them to perform the equivalent of an interactive user login. In this way, they can establish their own login context rather than running under the identity of the principal that started them.
- Controls the authorization that principals have to network resources in the DCE cell. Each object in the DCE cell can have an associated Access Control List (ACL) that specifies which operations can be performed by which users. ACLs can be associated with files, directories, and registry objects, and can be implemented by arbitrary applications to control access to their internal objects.

The DCE Security Service is implemented as a security server, which maintains a store of security information about network resources in its security database (also known as the DCE registry database).

Distributed Time Service (DTS)

To compensate for natural drifts in system clocks, the DCE distributed time service (DTS) ensures that all system clocks of the servers in a DCE cell are synchronized. This is especially important where servers are in different time zones. A time service is also essential to ensure the reliable operation of authentication and authorization services.

IBM WebSphere Transactional Integrity

WAS EE enables scalable, distributed transaction systems with security, reliability, availability and data integrity. The transactional features of WAS EE are offered in traditional procedural (CICS, Encina) and component-based (CORBA, EJB) programming models and related transaction processing (TP) monitors. It includes connectivity through a suite of server and connectivity options, including:

- CICS, CORBA and Encina clients for client connectivity across differing platforms.
- CICS Transaction Gateway and Transarc DE-Light Client and Gateway for integration between Java applets or servlets executing business logic and CICS or Encina applications managing resources in the transaction monitor. The CICS Transaction Gateway also provides a browser-enabled client for accessing 3270-based applications through a common Web browser.
- MQSeries for business quality messaging to integrate scalable distributed heterogeneous applications. MQSeries servers and clients enable CICS and Encina servers to communicate with each other through asynchronous messaging and queuing, and enable integration with other stand-alone MQSeries-based applications.
- CB Application Adapters for access to DB2 Universal Database, Oracle, CICS, Information Management System (IMS) or MQSeries-based applications. Manage object technology.

WAS EE provides a scalable runtime environment for developing and deploying distributed component-based solutions—including Enterprise JavaBeans. The component features of WAS EE allow networked applications to use, coordinate, store, translate and sort information from disparate existing or new applications and back-end systems. Its features include:

- A programming model that allows data access to be partitioned from business logic
- A CORBA 2.0 compliant ORB that uses the widely accepted IIOP standard to communicate with other complying ORBs
- An application runtime environment for the integration and management of object services
- Management of distributed application interactions with networked computing hardware and software resources (for monitoring, resource allocation, unit of work)
- Support for Web (Java) clients, traditional CORBA clients, ActiveX clients and an ever-increasing number of nontraditional clients, including kiosks, ATMs and others
- Multi-tier visual development tools for the major object-oriented programming languages

Application Server Architecture

The IA Runtime Topology is based on the Enterprise Solution Structure (ESS) Thin Client Transactional Pattern and is a representative solution for the SFA pattern. The pattern represents a starting point for extending business to the Web. In the initial release, there is no

interaction with legacy back-end systems (and Web pages are served by a single Web server). In the subsequent release, there are connections to legacy back-end systems.

There are several options to be considered when determining a runtime topology. Some of these options have implications on the runtime performance of the application. Other options will determine whether or not the application will have failover capabilities. Yet another option will allow for a secure connection to WAS EE.

The topology options are discussed in the following sub-sections, each describing advantages and disadvantages. The options appear in no particular order.

Separation of Web Servers from Application Servers

Under certain circumstances, you will have to separate the Web Server component from the Application Server component. This may be necessary when you want to put the Application Server behind a DMZ so it is in a secure environment. The corresponding Web Server can be placed inside the DMZ behind a firewall. The separation of the servers may also be necessary when you have a high volume site with limited business logic. In this case, there is no need to install an Application Server with each of the Web Servers.

This separation of the Web Server from the Application Server is done by means of the servlet redirector. The redirector is a stripped down version of the WAS EE that forwards requests to a dedicated server. There are two version of the redirector available, the Thick Servlet Redirector and the Thin Servlet Redirector. The Thick Redirector is required for the initial release.

Thick Servlet Redirector

The thick redirector runs as part of the administrative server. It has therefore the overhead of requiring the administrative server's infrastructure, e.g., the database. The advantage of the thick redirector is that it can be configured and managed using the WebSphere Administrative Console from either the redirector's or application server's node. Also, the thick redirector is capable to redirect to secured resources, whereas the thin redirector can not do this.

Thin Servlet Redirector

The thin redirector does not require the administrative server's infrastructure and does therefore not suffer from any overhead. It is especially suitable when you have limited resources or do not want to expose any information in the database. Its disadvantage is that it cannot be configured and managed using the Administrative Console. All configuration is done by means of scripts that need to be executed on a regular basis. This means that the redirector can be out of sync whenever the Application Server has changed and the changes have not been propagated to the Web Servers. Also, the thin redirector can not redirect to secured resources, whereas the thick redirector can do this. It depends on your requirements as to which version is a feasible alternative.

If access to secured resources is not an issue, either one can be used to separate the Web Server from the application Server. If, however, resources are secured, then only the thick redirector is an option.

Since the requests to the actual Application Server need to be forwarded, there is a slight performance degradation to be expected. In addition, a firewall needs to be configured. But this firewall delivers additional security.

The Number Of Web Servers

The number of Web Servers is crucial when it comes to serving high volume sites. This is because each Web Server can serve only a limited amount of clients at the same time. As soon this limit is reached, additional users who want to access that site will receive error messages that the site cannot be reached. But this information is incorrect because the Web Server is connected. It just cannot serve the requests.

To address this performance issue additional Web Servers may be installed. Each of these Web Servers will be connected to the corresponding Application Server. The number of Application Servers can be the same as the number of Web Servers or it can be smaller, i.e., some Application Servers will be connected to more than just one Web Server. However, a Web Server can connect to only one Application Server.

The ability to add additional Web Servers when needed implements, horizontal scaling. It also increases the site's availability.

The Number Of Application Servers

Another means to increase the availability of a site is to increase the number of Application Servers that are installed. Increasing the number of Application Servers increases the failover capabilities as well as the performance of the application. This is particularly the case when the business logic is implemented in the Application Servers and little or no business logic is implemented in the backend systems.

Increasing the number of application servers does not necessarily mean that the overall throughput will be increased as well. The throughput will not increase if the backend systems have reached their limits. Overall, it is highly application-dependent as to whether or not a performance improvement can be achieved by introducing additional Application Servers.

Clones Running On Application Servers

In order to increase the vertical scalability of an Application Server, the number of processors installed on that particular server can be increased. Each of these processors can then execute in parallel. However, a performance improvement is only possible if either the software or the operating system can make use of these additional resources.

The WAS leverages additional processors by means of "cloning" of applications. Cloning means that a given application is duplicated such that the clients cannot distinguish between the clones. In addition, each WAS is running on its own Java Virtual Machine (JVM).

The Application Server balances the workload of the clones running on a server automatically. Thus, no administration is required to manage machine utilization.

There are circumstances where cloning is desirable even if there is only one processor installed. This can be the case if there is an application that is spending most of its time waiting for some resources. During this time additional requests can be served by the application clones. Also, if automatic tasks like Garbage Collection take too long to complete, cloning may prove a viable alternative on a single-processor machine.

In addition to all the before mentioned advantages, cloning also increases the availability of a particular application as well as the failover capability or if any of the clones fail, the other clones take over the workload. Overall, it is application-dependent as to whether or not a performance improvement can be achieved by cloning of applications.

As with all alternatives there are also disadvantages to consider:

- If you do not require session affinity and want all session-related information to be transparent to the users, there is a performance impact because all session information needs to be saved and retrieved from a database.

- If your application assumes that it is running on a dedicated machine (even on some dedicated JVM), cloning will not be an issue for you because it cannot be determined a priori on which machine you application will execute the next time a request is served.

10.6. Component Broker

The Component Broker component is the IBM ComponentBroker COTS product that implements CORBA. This component provides object-based functionality to the Application Server component. An object-based capability provides a standard model for object state and behavior accessible through object methods.

CB is an enterprise solution for distributed computing, providing a scalable, manageable environment for developing and deploying distributed component-based solutions. It is an integrated implementations of the open standards contained in the OMG CORBA initiative. Many of the low-level details of the CORBA interface are hidden by the CB's easy-to-use framework.

Primarily, CB is an object server. It comes with a development environment that is optimized for creating business objects that run in the CB server. This server consists of both a run-time package called the CB Connector (CBConnector) and a development environment called VisualAge Component Development for WAS V3.0 Enterprise Edition (CB Tools). The run-time package provides a server in which business object components run and are managed through a set of management tools.

The CB run-time environment supports the execution of C++ and Java-based business logic that follows the CORBA model or the model of the EJB specification from Sun Microsystems. The EJB specification provides a portable, platform-independent reusable component architecture. These components run in a robust multi-threaded server that provides easy access to a wide variety of services and capabilities. Key object services provided include:

- Data Cache and Prefetch
- Concurrency Control
- Event
- Externalization
- Object Identity
- LifeCycle
- Naming
- Query
- Security
- Transaction

These services are available in an integrated fashion based on the OMG model and the CB Managed Object Framework (MOFW). This framework exists as a set of configurable and extensible interfaces that are adapted to meet specific requirements.

Application adaptors extend and specialize these interfaces. Application adaptors provide a home and container for Managed Objects, similar to an object-oriented database. Combinations of these object services are called qualities-of-service and are acquired according to the containers in which the objects run. Developers build the business logic, and the administrators make decisions about the qualities-of-service to be provided for any given installation of a business object.

CB contains a CORBA–2.0-compliant ORB that is encapsulated by the MOFW and the object services provided by CB. The ORB facilitates interoperability with other IIOP servers and with clients. CB supports client access by using the IIOP directly or by using RMI over IIOP (RMI / IIOP).

Many business object abstractions depend on existing resources. CB supports the separation of the business logic from the state data in the base programming model. This separation is managed and controlled by the CB run time. CB provides access to resources on DB2, Oracle, CICS, IMS, and MQSeries. Business objects that have data objects backed by these resource managers can participate in distributed transactions. CB acts as an external commit coordinator for all of these resource managers through the implementation of the CORBA Object Transaction Service (OTS).

System Management tools enable administrators to control the distributed-object computing environment. These tools enable the modeling of the deployment configuration and then the actual management of the abstractions that are introduced by the distributed-object paradigm. Administrators can enlarge configurations by adding servers and server groups.

They can alter qualities-of-service through container management, and they can scale up environments by adding additional computing resources into the object server pool.

Development of applications that run on CB can be done in one of two ways. First, CB provides a complete set of tools and support for building CORBA-based applications. Second, CB tools enable the deployment of enterprise beans that were created by using other tools, such as VisualAge for Java. A CORBA-based application is developed by using the Object Builder tools that come with CB. Designs for systems can be imported from the Rational Rose ® visual-modeling tool. After these designs are imported into Object Builder, the template for the implementation is available for use.

Developers merely fill in the business logic in prescribed places within the framework. Object Builder takes care of the rest. It generates the code, makefiles, and application configuration information necessary to test the application on a CB server. Object Builder also supports large-scale team development. Object Builder facilitates transformation of large-system object architecture into separate pieces that can be worked on by many teams. Object Builder then facilitates linking these models back together and building an integrated solution.

Object Builder also serves as the deployment tool for enterprise beans that run in the CB run time. Object Builder facilitates the mapping of entity beans with Container-Managed Persistence (CMP) to databases and existing applications. Distributed object applications that leverage both enterprise beans and CORBA-based objects are easily constructed, tested, and deployed using the Object Builder tool and the CB run time.

CB is also an EJB server environment. It supports deploying and running components based on the EJB 1.0 Specification. It provides the qualities-of-service prescribed by the EJB specification and allows developers to build enterprise application business objects using the EJB programming model. Support for CMP allows mapping to a wide variety of resource managers. This support is based on the same technology used to map CORBA-based business objects to existing resource managers. Enterprise beans also benefit from the same implementations of object services that are available to CORBA-based business objects.

Component Broker Application Architecture

CB applications are designed as three-tiered applications. The content of each tier is summarized below:

- First tier—A programming model that allows client applications to be implemented in C++, Java, or Visual Basic, and allows clients to access components on the server through a CORBA-compliant ORB.
- Middle tier—A programming model with full tools support, and a run-time environment, in which the components (CORBA-based business objects or enterprise beans) are deployed.
- Third tier—Application adaptors on the middle tier allow components to access data in various resource managers, including DB2, Oracle, CICS, and IMS.

Three-tiered applications are reliable, extensible, and scalable. The MOFW makes the development of such applications possible. The framework is supported by a suite of development tools, which allow you to make use of the framework to create components without going into details of inheritance and framework implementation.

CB provides components that are deployed in the middle tier, connecting the first tier (client) with the third tier (databases and other resources). The components are implemented as CORBA-based business objects or enterprise beans. They allow application logic to run on high-powered servers, and insulate client applications from the complexities of the various resource managers. The client works with the components through a CORBA-compliant ORB, and the components work with the resource managers.

In combination with existing resource managers, CB functions as an object server by providing an application environment that lets clients access back-end-systems through object-oriented middleware. This system provides an infrastructure scalable enough to include everything from desktops to the largest cluster of mainframes. CB is platform-independent and allows design, development, and deployment of distributed object-oriented server applications for mission-critical solutions.

A CB server process provides a complete execution environment and partial implementation for managed objects. Method requests are routed from the ORB to the server. The server, in conjunction with the container and adaptor, ensures that method requests are dispatched on a properly prepared object. This means that the object may have to be activated in memory, its state data may need to be refreshed, or it may need other services attached to the request before execution. Some examples of additional services provided to managed objects follow:

- Persistence, transactions, and security
- Workload management and availability management over multiple servers
- Object-oriented access to existing databases and applications

CB enables operational reuse. Operational reuse focuses on reengineering existing software so it can be used as building blocks for new applications. The CB infrastructure enables the construction of these new building blocks. These building blocks achieve operational reuse and present an abstraction layer that can in turn be reused to build many new business applications.

CB clients can be C++ programs, Java applets, or Visual Basic programs that a user interacts with directly, or they can consist of Web Servers or Application Servers that have their own clients. A typical client application consists of view objects, which provide the end-user interface interactions and a mapping to server components, and client objects, which implement business logic specific to the client and provide integration with other desktop applications, such as spreadsheets and document processors. Anything that can send an IIOP request is a potential client.

The server can have components on one machine or on many different machines. This layer includes programs written by developers as well as CB components that make up the infrastructure. The third tier can be running on many different physical hosts, and can be

using application logic that itself provides additional physical tiers. In the first tier, clients access CB server objects through proxy objects.

Middle-tier architecture

The middle tier of the three-tier architecture consists of components implemented as CORBA-based business objects and enterprise beans. The architecture for the middle tier of a CB application has three layers, with three corresponding kinds of components: persistent components, composite components, and application components.

Persistent components are abstractions that map to existing data or procedural programs. While these components can have dependencies on other persistent components (one-to-one and one-to-many relationships), they are generally fine grained enough to be highly reusable.

Composite components represent new abstractions that are easily usable and understandable by client programmers and by application component programmers. These compositions often represent an aggregation of persistent components. Methods of the composite component typically delegate or bridge down to methods on the persistent components, which the component aggregates. These mappings can be one-to-one (mapping directly to a method of an aggregated component) or one-to-many (executing methods on each of the aggregated components). While composite components are not as reusable as persistent components, they can be more valuable when they are reused, because they represent larger portions of the application.

Application components focus on business logic and usage of other components. This is the layer that is accessed by client applications. Application components implement processes or tasks, as defined by object-oriented analysis and design. They implement any business logic that is not properly modeled as a method on other, individual components. Application components also provide the mechanism for moving application logic from the client to the server. Generally, application components represent the parts of the application architecture that are specific to an application, such as certain business processes or tasks. By separating out these elements, which are less suitable for reuse, leave the rest of the application (composite components and persistent components) as reusable as possible.

When extending an existing CB application, or creating a new application that uses the same data, create new application components to provide application-specific business logic. However, it is possible to reuse the persistent components and potentially the composite components. The persistent components need to change only if the underlying data store changes. The composite components need to change only if the definition of the aggregation changes. Even when change is necessary, the underlying component architecture allows the data store to change without affecting reuse of a component's behavior or interface.

Component Architecture

Each component consists of a set of objects, which work together and are managed by an application adapter. The component acts as a single object, even though it is implemented as a set of objects on the server. The client accesses the component to perform business

functions. All CB server objects are derived from the MOFW, which provides default code that allows the set of objects to work together and behave as a single entity. Components are factored into multiple objects to make them more maintainable. A component mapping to a data store is kept in a separate object, so if the mapping changes or a different data store is chosen, only the separate object needs to be updated. Application components and composite components are made up of three basic objects, along with some helper objects for locating and creating the component. Persistent components add a fourth basic object, which provides the code to access resource managers or data stores on the third tier. The main component objects are:

- Business Objects
- Managed Objects
- Data Objects
- Key and Copy Helpers
- Persistent Objects

Business Objects

Business objects define the interface of the component, in terms of attributes and methods. The CORBA Interface Definition Language (IDL) is used to define a business object's interface. IDL specifies an object's interfaces, independent of operating system and programming language.

A business object can be implemented in either C++ or Java. Because the business object is derived from the Managed Object Framework, the only code needed to provide is the implementation for any application-specific methods defined. When you create a business object using CB tools, the framework is extended. Attributes considered as part of the state of the object can be cached or delegated to the data object.

Data Objects

Data objects manage the persistence of component state information. They provide an interface for the business object to get and set state data.

A data object isolates its business object from having to:

- Know which of many data stores to use to make its state persistent
- Know how to access the data store
- Manage access to the data store

Managed Objects

Managed objects provide the component with management by an application adapter. Because this management capability is provided in a separate object, the type of service provided can be changed without affecting the component interface. Some examples of managed services are:

- Persistence, transaction, and security
- Workload management and availability management over multiple servers
- Object-oriented access to existing databases and applications

Keys and Copy Helpers

A component key object defines which attributes are to be used to find a particular instance of the component on the server. The key consists of one or more of the business object attributes, which must contain enough information to uniquely identify an instance.

A copy helper is an optional object that provides an efficient way for the client application to create new instances of the component on the server. The copy helper contains the same attributes as the business object, or a subset of them. Without a copy helper, the client might need to make many calls to the server for each new instance: one call to create the instance, and then an additional call to initialize each of the instance's attributes. With a copy helper, the client can create a local instance of the copy helper, set values for its attributes, and then create the server component and initialize its attributes in one call by passing it the copy helper.

Persistent Objects

A persistent object is an object that provides a mechanism for storing a component's state in a data store. Every persistent object has an identifier or a key that is used for locating its corresponding record within the data store. There are two main kinds of persistent objects:

- Those used for accessing database (DB) data (mapping to a DB schema)
- Those used for accessing procedural data (mapping to a CICS and IMS bean, or Procedural Adaptor bean).

Component Instantiation and Execution

Each component is instantiated and later located by using a home object. A home object is a server object that allows clients and other components to locate and create components of a certain type. Once created, the components live in a container, which acts as an application adapter for the component, providing management services to the component through its managed object. When a call is made to a persistent component get method (to retrieve the value of an attribute) the component objects work together as follows:

- The managed object accepts the call, and calls its associated container for object services before passing the call on to the business object
- The business object accepts the call, and either returns the value of the attribute based on a cached copy of the data or delegates the call to the data object
- The data object accepts the call, and either returns the value of the attribute based on a cached copy of the data or delegates the call to the persistent object

Common Object Request Broker Architecture (CORBA)

The OMG created the CORBA specification to facilitate the development of object-oriented applications across a network. CORBA objects are standard software objects implemented in any object-oriented programming language. An ORB mediates the transfer of messages between client programs and objects. When a client program invokes a method on an object, the ORB intercepts the request and finds an object implementing that method. The ORB returns the result of the method invocation to the client program. From the programmer's point of view, all of the work appears to be done on one computer system.

IIOP enables communications between different ORB implementations. The IIOP is based on TCP/IP and includes additional message-exchange protocols defined by CORBA.

This section discusses the standard parts of an ORB used in WAS EE.

CORBA IDL

One of the most important parts of the CORBA specification is the IDL. This object-oriented language enables programmers to create interfaces between components written in different programming languages, so that objects can be used by remote clients as if the objects were local to the client. This interaction between language-disparate components provides great flexibility to developers working in environments with many platforms and development tools.

The component interface is created in a CORBA IDL file, which is then compiled by using the CORBA IDL compiler. This produces most of the stub and skeleton files required for creating a distributed application.

Remote calls

CORBA remote calls enable communications between client proxy objects and server-side objects. CORBA servers export implementation objects to which client proxy objects bind in order to obtain a service. Objects that participate in transactions or make transactional requests on other objects are called transactional objects.

A remote call in CORBA occurs when a client creates a proxy object and uses that object to invoke a method on a corresponding implementation object at the server. This remote call is similar to an RPC.

Naming and Binding

The CORBA services specification details the manner in which a naming service must be implemented; however, the specification allows for differences between ORBs. Whether a particular ORB contains a naming service is left to the discretion of the ORB implementer. Both Orbix and the CB ORB contain a naming service.

ORBs that do not implement a naming service must implement some way of enabling a client to bind to a server. For example, the Orbix _bind function can be used instead of the Orbix Naming Service.

Component Object Model (COM)

Microsoft COM is a model for developing applications composed of individual components that can be transparently and separately upgraded. A COM component can be used to create an application in any language. Once instantiated in a client, a COM object acts as a proxy for the client, marshaling and unmarshaling data to contact a server and returning data and status information back to the client. Both TXSeries and CB provide COM functionality.

10.7. Content Management

The Content Management component is implemented with Interwoven TeamSite and Interwoven TeamSite Templating. The SFA application developers and content authors to manage the SFA Internet and Intranet Website use this COTS product.

The Interwoven TeamSite COTS product enables Web developers to work in an environment that supports versioning of file system and database assets. Control of the assets is accomplished by using an easy-to-use workflow engine. The product utilizes branch structures to organize workgroups and enforce security, provides developers with their own Workarea or sandbox to develop within, and will store all versions of the Website.

The TeamSite Templating COTS product is an add-on package that enables decentralized content contribution, even in environments with centralized control of the overall look and feel. The product:

- Give users the ability to submit content that can be stored directly in a database or a file, while preserving the integrity of Web page layout site architecture,
- Ability to create dynamic Websites. Common content elements can be combined with data stored in a database or file system to deliver dynamic Web pages,
- Enables content reuse by providing the ability to include individual content elements in any number of Web pages, and
- Administrators can implement a consistent look and feel across the Website, while maintaining the flexibility needed to make rapid changes.

The content will be controlled through a workflow engine. Once updates are approved to go to the test / development machines, the data will be transferred over to the storage area network (SAN). Viador and WebSphere access the SAN in order to display content through the browser.

After the Web content has been tested and approved in the development environment, it will be transferred to a different location on the SAN in production. The production servers will access the content in the same manor as in development. An external task may be launched at this time in order to alert Autonomy that content has changed.

The TeamSite Intelligent File System is composed of:

- The TeamSite server and kernel
- The TeamSite backing store of files and metadata
- A suite of command-line tools
- The TeamSite CGI
- Proxy servers for access through the TeamSite browser-based graphical user interface (GUI)
- File system mounts for access through the file system interface

The Intelligent File System is the core of the TeamSite system, where detailed information about the Website, the Web assets, Web asset metadata, the production process and the users is stored. The Intelligent File System collects and maintains metadata on TeamSite files, directories, and areas, and allows TeamSite to process and present information according to who is asking for the information, and under what conditions. By using an object-oriented design within a file system architecture, TeamSite combines extensive meta-data tagging with open access and file system performance for Web content.

The following diagram illustrates the elements of the SFA TeamSite deployment.

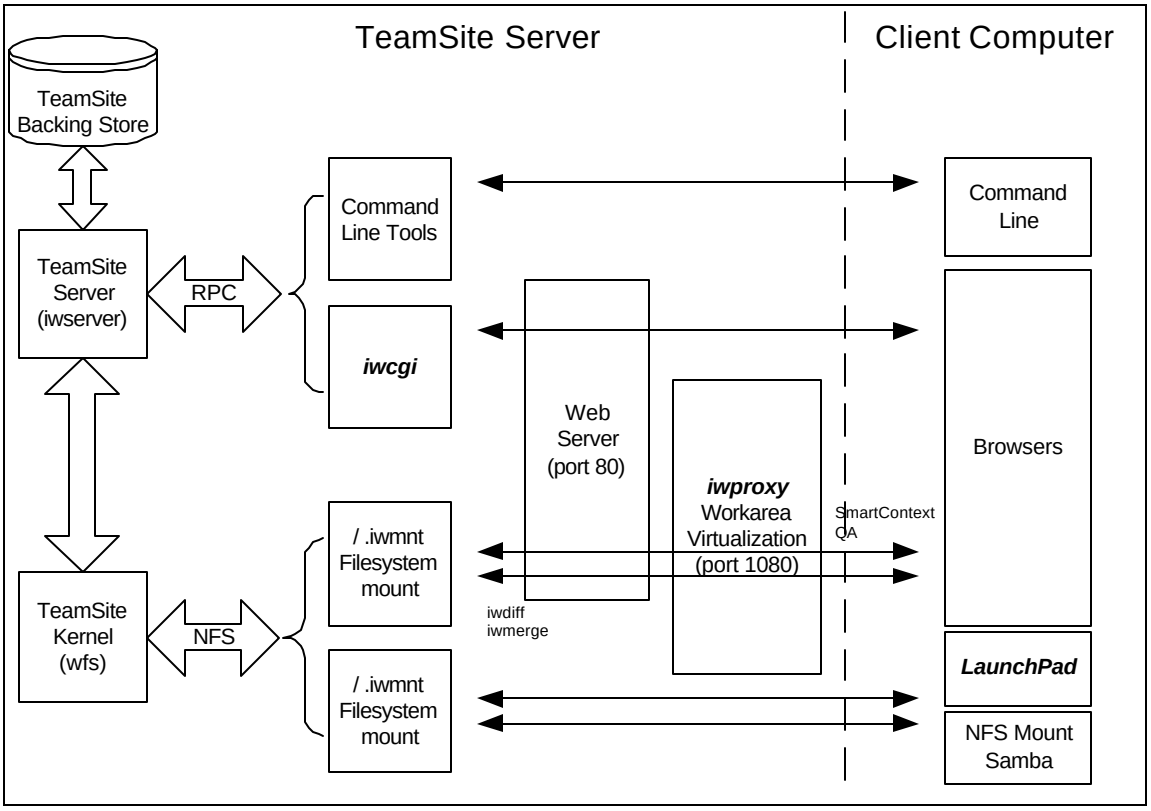

Figure 27 – TeamSite Server

The client computer connects to the TeamSite server in several ways. Requests from the browsers or LaunchPad are routed through the standard TeamSite proxy server, which

allows consistent views of TeamSite areas. The double proxy server redirects hard-coded links within the Website. Requests through the File Storage component (AFS) and command-line tools, which do not go through the Web Server component, are not routed through a proxy server.

10.7.1. Private Workareas

With TeamSite, each Web contributor has a private workarea based on the server to develop and modify Web content. Each workarea contains a virtual copy of the entire Website. This allows each Web developer to stage and test changes in the context of the entire site without impacting the Web site or other contributor.

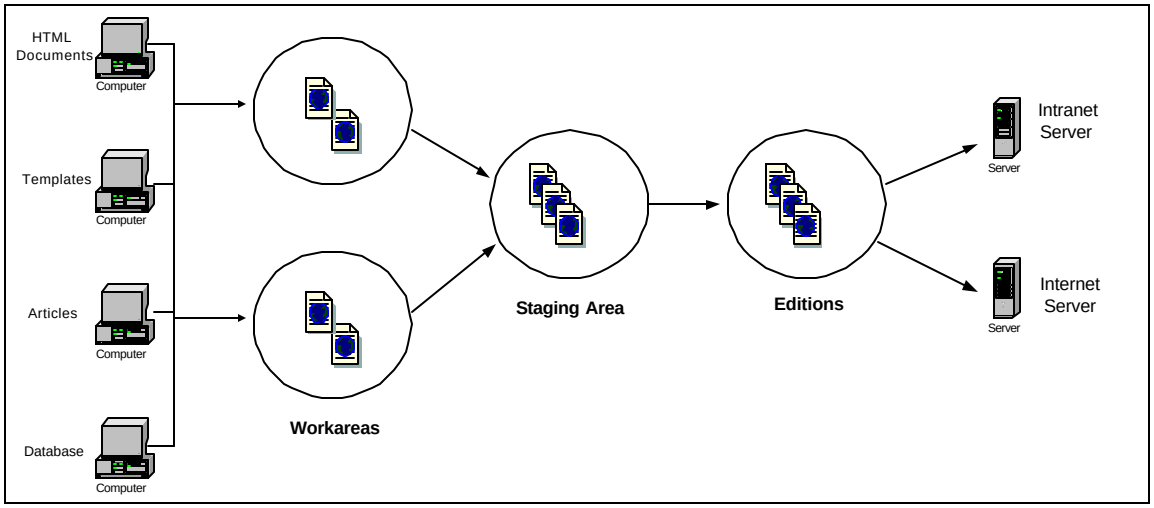

Figure 28 – TeamSite Workflow

Workflows can follow many different paths, but the one depicted above is the most likely simplest scenario to demonstrate content management and how it is deployed to the respective Websites. Web contributors for the SFA Intranet and Internet will place their content into a Workarea that is created by an administrator. Once the content, for example an HTML page, is completed, the programmer will submit it to staging. Once in staging it can go through an approval process before being placed onto the next edition. Editions are moved onto the actual Websites either during scheduled or event driven processes.

10.7.2. TeamSite Elements

The following sections describe the TeamSite elements. These elements consist of Branches, Workareas, Staging Areas, and Editions.

Branches

TeamSite provides branches for different paths of development for a Website. Branches can be related to each other (e.g. alternate language versions of the same Website) or they may be completely independent. Each branch contains all the content for a Website.

A single branch contains archived copies of the Website as editions, a staging area for content integration, and individual workareas where users may develop content without disturbing one another. Branches can also contain sub-branches, so teams may keep alternate paths of development separate from each other. Content can be easily shared and synchronized across branches and sub-branches. Users may work on one branch or on several, and the number of branches on a system is not limited.

Branches facilitate distributed workflow because they allow separate teams to work independently on different projects. Because all branches are located on the same TeamSite server, it is easy for one team to incorporate the work of another into their project.

Workareas

Each workarea contains a virtual copy of the entire Website, which may be modified in any way without affecting the work of other contributors. Users who have access to a workarea may modify files within that workarea and view their changes within the context of the entire Website before integrating their work with that of other contributors. Users can lock files in each workarea, eliminating the possibility of conflicting edits.

All changes that are made to files in a workarea are kept completely separate from other workareas and the staging area until the user chooses to promote his changes to the staging area. Within a workarea, users may add, edit, or delete files, or revert to older versions of files without affecting other users.

Staging Areas

Each branch contains one staging area where contributors incorporate their changes with the work of others. Users submit files from their workareas to the staging area to integrate their work with other contributions, and test the integrity of the resulting Website. Because the staging area is an integrated component of the system, conflicts are easily identified and different versions of the same file can be merged, rather than overwritten.

Editions

Editions are read-only snapshots of the entire Website, taken at sequential points in its development. Contributors can create new editions any time they feel their work is well integrated, or any time they want to create an update to the Website for reference or deployment. Each edition is a fully functional version of the Website, so the users may see the development of the Website over time and compare it with current work.

10.7.3. TeamSite Users

The following sections describe the TeamSite users. These users consist of Authors, Editors, Administrators, and Masters.

Authors

Authors are primary content creators. All work done by Authors goes through an explicit approval step. They can receive assignments from Editors, which are displayed in To-Do lists when Authors log in to TeamSite. Authors can access TeamSite from a Web browser.

In order to test work, Authors have full access to the content in their Editors workareas, but do not need to concern themselves with the larger structure and functionality of TeamSite. The Author role is appropriate for non-technical users, or for more technical contributors who do not need access to the TeamSite extended functionality, such as the TeamSite advanced version management features.

Editors

Editors own workareas. They create and edit content, just as Authors do, but they are primarily responsible for managing the development taking place within their workareas. This includes assigning files to Authors and submitting completed content to the staging area, and it may include publishing editions.

Editors have access to specialized TeamSite content and workflow management functions. Editors are generally "managerial" users, who primarily supervise the work of Authors, or self-managing "power" users, who need the TeamSite extended functionality to manage their own content.

Administrators

Administrators own branches. They have all the abilities of Editors, but they are primarily responsible for the content and functioning of their branch. Administrators can manage project workflow by creating new workareas for Editors and groups, and by creating sub-branches of their own branch to explore separate paths of development.

An Administrator is the supervisor of the project being developed on his branch. The administrator may be the Webmaster for a particular version of the Website, or a project manager.

Masters

Master users own the Website. They can perform all the functions of Editors and Administrators on any branch. The Master user owns the main branch, from which all sub-branches are created. The Master user is generally involved in the installation of TeamSite, and can reconfigure TeamSite on a system-wide basis.

10.7.4. TeamSite Templating Model

Developing new templates requires a strict file and directory structure. TeamSite Templating uses a data storage hierarchy based on data categories and types. The directory structure supporting this hierarchy resides in the workarea for each TeamSite Templating user. The

directory structure is illustrated in the following diagram. Items in boxes are directories; items not in boxes are files.

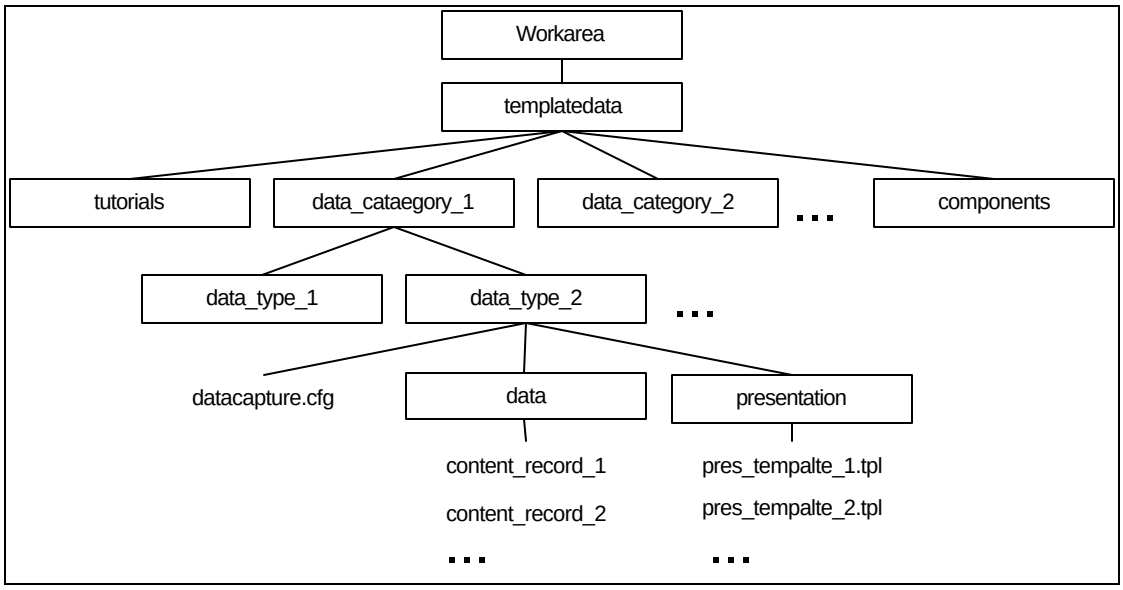

Figure 29 - The templatedata directory is at the highest level in the hierarchy.

Data categories are at the next level in the hierarchy and contain one or more data types. In addition to residing in this directory structure, data categories and types must also be listed in the templating.cfg configuration file to be made available to TeamSite Templating. The component directory that stores component templates is also a subdirectory of templatedata.

Data type directories each contain a datacapture.cfg file and the subdirectories data and presentation. Details for the entire hierarchy are shown in the following table.

Table 45 – Datacapture.cfg Directory Hierarchy

File or Directory	Description
templatedata	Top-level directory containing subdirectories for data dategories, types, and all associated configuration files. Residesin the workarea for each user who uses TeamSite Templating. Can be renamed.
data_category_1	The first major categorization for data on a specific branch. Named and defined in templating.cfg. For example: /templatedata/beverages
data_type_2	The first subcategory of data in data_category_1. Named and defined in templating.cfg. For example: /templatedata/beverages/tea Each data type in a given data category has its own subdirectory.

File or Directory	Description
datacapture.cfg	The XML configuration file that defines a data capture template and drives data capture for a specific data type. As such, it defines the data type itself For example: What information the data type will contain, parameters for what type of data is legal in any input field, etc.) Specifies the look and fell of the data capture form displayed in the TeamSite GUI through which a content contributor enters data. Each data type must have exactly one datacapture.cfg file.
data	The directory containing all captured data content records for a given data type. If necessary, you can define and create a directory tree underneath the data directory. A data directory can contain zero or more data content records.
content_record_1	The first data content record for a given data type. Each data content record is an XML file containing formatting information interspersed with data that was captured from a content contributor via the TeamSite GUI. A data content record is named by the content contributor during data entry. For example: /templatedata/beverages/tea/data/november_order
presentation	The directory containing all presentation templates for a given data type. The presentation directory must contain one or more presentation templates.
pres_template_1.tpl	The first presentation template for a given data type. A data type can have any number of presentation templates. A single presentation template is populated by data from zero or one data content records. A presentation template can have a name of your choice. For example: /templatedata/beverages/tea/presentation/monthly_order.tpl
components	The directory where all component templates are stored.
tutorials	Examples showing the use of ix_xml tags.
data_type_2	A second subcategory of data in data_category_1. For example: /templatedata/beverages/coffee
data_category_2	A second major categorization for data on a specific branch. For example: /templatedata/food

The TeamSite Templating architecture allows data capture and data presentation to be configured, executed, and managed separately. The following diagram and sections provide a high-level overview of this architecture:

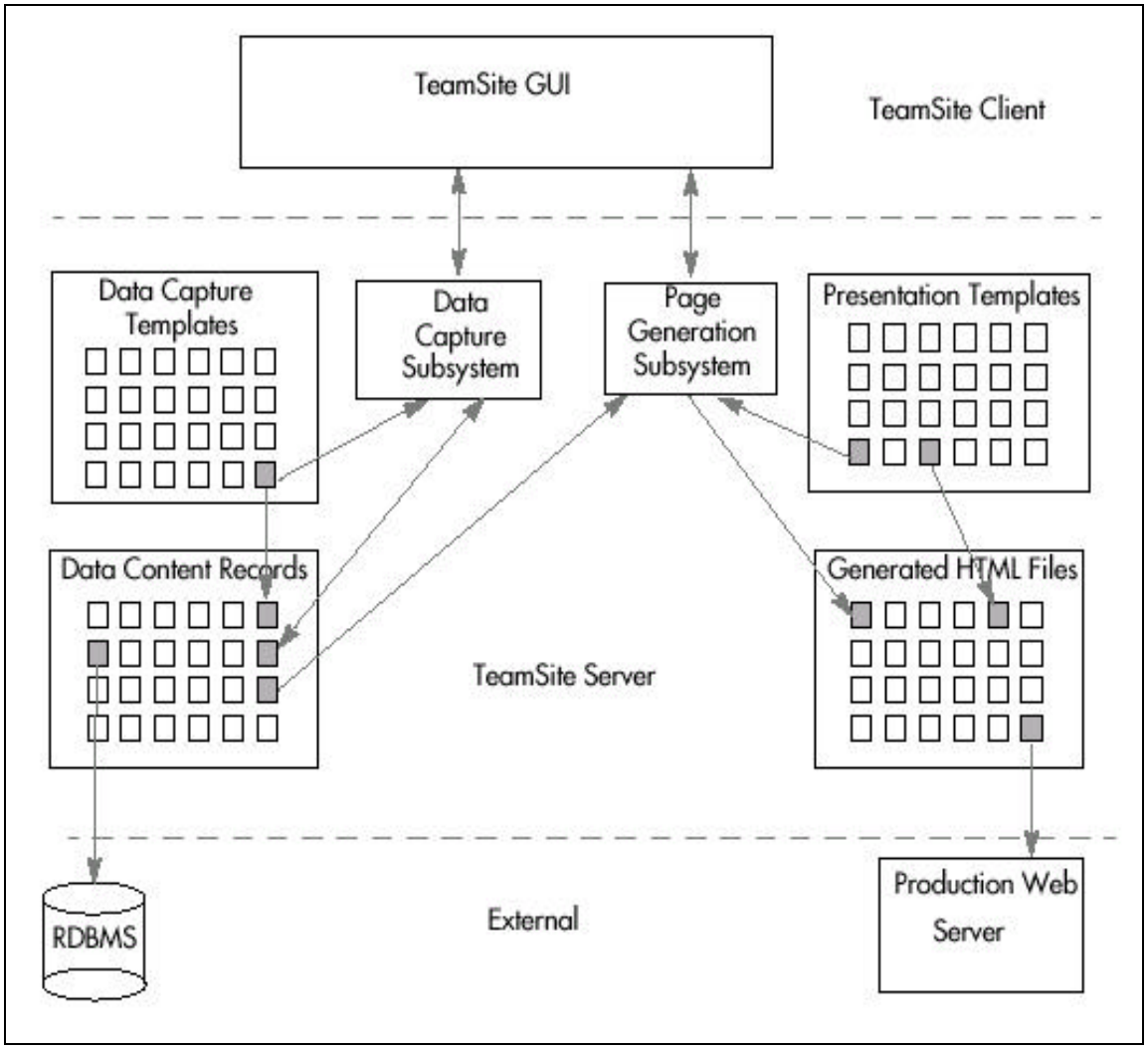

Figure 30 –TeamSite Templating Overview

Data Capture

Content contributors working through the TeamSite GUI have access to the data capture subsystem. This subsystem lets content contributors select and work through forms defined by data capture templates to create or edit data content records, which by default are stored in the TeamSite file system. After data content records are created, they can be displayed via presentation templates or optionally deployed to a database via DataDeploy.

Data Presentation

After data is captured and stored as data content records, users working through the TeamSite GUI or from the command line can access the page generation subsystem to combine a data content record with a presentation template. The end result is a generated HTML file that displays the data content in a way defined by the presentation template. Additionally, users can generate an HTML file that obtains data from zero or one data

content records and from queries to databases. The generated HTML file can optionally be deployed to a production Web server via OpenDeploy.

10.7.5. Interwoven Production Topology

TeamSite is installed on Sun Enterprise 3500 with Solaris 2.6. The IHS Web Server is installed with the TeamSite server. The following Interwoven products are deployed.

TeamSite

TeamSite is deployed with sample content which will be used to test and demonstrate proper installation and configuration. There is a requirement to have a directory iw-store to be in a separate partition. The iw-store is an Intelligent File System that is composed of:

- TeamSite server and kernel
- TeamSite backing store of files and metadata
- A suite of command-line tools
- TeamSite CGI
- Proxy servers for access through the TeamSite browser-based GUI
- File system mounts for access through the file system interface

The Intelligent File System is the core of the TeamSite system and manages detailed information about the Website, the Web assets, Web asset metadata, the production process and the users. The Intelligent File System collects and maintains metadata on TeamSite files, directories, and areas, and allows TeamSite to process and present information according to the client that requests the information, and under what conditions. By using an object-oriented design within the File Storage architecture, TeamSite combines extensive metadata tagging with open access for Web content.

Templating

TeamSite Templating provides a highly configurable way to capture, edit, and store data input from content contributors; define the appearance of displayed data; and integrate captured data with other Interwoven products such as TeamSite Workflow and DataDeploy. The TeamSite Templating mechanism for capturing data content from content contributors is separate from the mechanism for defining the appearance of the content when it is displayed. This architecture allows for unlimited reuse of data after the data is captured and stored. It supports different appearances and behaviors for the same data content based on how, when, where, or to whom the data is displayed.

OpenDeploy

The OpenDeploy utility is used to transfer the Schools Portal and associated applications Website content from the development environment to the production environment. It supports cross-platform deployment from the development servers to any Solaris or

Windows NT production servers (and vice versa), as well as allowing Solaris-to-Solaris or Windows NT-to-Windows NT deployments.

OpenDeploy allows transactional deployments, so that if deployment is interrupted, the SFA Website content that was originally on the production server will remain intact. OpenDeploy also allows SFA administration to specify the degree of detail to be included in the deployment log files from a broad overview of the deployment to detailed information about each file that was deployed.

OpenDeploy server is installed on the WebSphere and Viador servers. Content will be transferred to the development machines after the Website content developer has submitted it. Once the content has been tested on the development environment servers it can then be deployed from the TeamSite server to the production servers.

DataDeploy

The DataDeploy utility is used to transfer extended attribute data between TeamSite, an external Structured Query Language (SQL) database (Oracle 8i), and an XML file. For the TeamSite-to-database scenario, the DataDeploy Database Auto-Synchronization (DAS) module is used to automate the entire deployment process for TeamSite Templating users. In this situation, any extended attribute changes resulting from modifying a data content record via the TeamSite Templating GUI are automatically deployed to a database.

DataDeploy will interface with the Oracle ODS in order to update the database once a user has entered content through the predefined templates.

How content will be deployed to production

Once content is approved in the staging area it will be copied to a different location on the SAN for testing. The testing environment will be a duplicate environment of the production environment. Once testing has completed an approved Editor or Administrator will promote content to an Edition. OpenDeploy will then copy the new Edition out to the production environment.

Autonomy Interface

When data is deployed, an external task can be called, and a content file is produced. After the content file is approved Autonomy would be launched via the script to spider the new content. Another scenario is to schedule the deploys so that the content is first deployed and a file written to that will be used by Autonomy to spider that content. Another alternative approach is to have no connectivity between Autonomy and TeamSite and just let Autonomy spider all the content.

Oracle Interface

In some cases it may be desirable to have metadata from the templates to be placed into a database. Since SFA is using a relational database management system (RDBMS) (Oracle)

this can be done through DataDeploy. Whenever a TeamSite-to-database deployment finishes executing, the end result is an updated table on the destination system. This table will be either a base table, delta table, or standalone table, depending on what type of update you instruct DataDeploy to perform (as defined in the configuration file's <update> section). Update types are named for the type of table they modify.

The most common sequence of events when deploying from TeamSite to a database is as follows:

- Generating an initial base table of a staging area or edition
- Generating a delta table for each workarea associated with the staging area or edition

It should be noted that an OpenDeploy server will need to be loaded on all machines that require content. OpenDeploy utilizes a particular port to another instance of OpenDeploy to insure data integrity.

The following diagram depicts how the data will flow from the TeamSite server to the production environment. How often this occurs will be determined by the development team. It is most likely that it will be a scheduled deployment once or twice a day. Since TeamSite is installed on a Solaris machine this is accomplished through a cron job. The cron job will invoke OpenDeploy to do the "copying" of data and possibly any manipulation that may be needed (for example changing owner and permissions of the files).

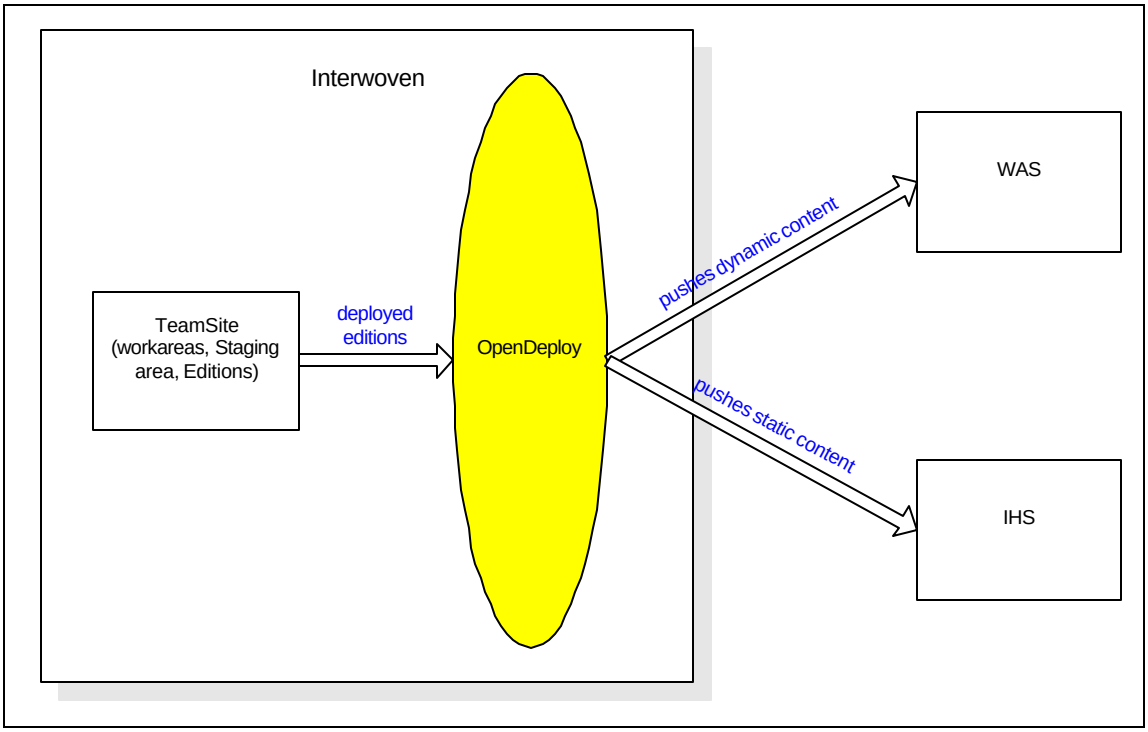

Figure 31 - TeamSite Content Dataflow

10.7.6. Security

Access to TeamSite is governed by two factors: File permissions and TeamSite access privileges. File permissions control access to individual files and directories. Password authentication is used when logging in to TeamSite. However, TeamSite access privileges govern log in under various roles, and access to branches and workareas. For example, to edit a file in a workarea, a user must both be able to access that workarea (through TeamSite access privileges), and have permissions for that file and its parent directory (through permissions).

When adding a new user, take the following three factors into account:

- Whether the user has access to the server
- The role the user will play in your Website operations
- The portion of the Website the user will be editing

To decide what groups the new user needs to belong to, and which workareas he needs to access, consider the existing groups and which portions of the Website and which workareas they can access. Add the new user to the groups that work on the same portion of the Website that he will be editing, and he will automatically have access to their workareas and to their Website files. If the new user needs his own workarea, create a private or shared workarea for him, but make sure that he owns or has group-level access to the files that he will be editing.

When creating a new workarea, the following needs to be decided:

- What the name of the workarea should be
- Who will need to access the workarea (this can be one person, or one person and a group)
- What portions of the Website the workarea's to which owner and group should and should not have access

Set permissions on your files according to the latter consideration. Permissions cannot be set differently for different workareas. If the permissions for corresponding files are set differently in different workareas, you will encounter conflicts when you submit files to the staging area.

It is useful to keep a chart of the Website that shows what users and groups have access to what sections of the Website.

Comprehensive Security

File Versioning

TeamSite robust file version histories are created transparently and easily accessed.

Whole Site Archiving

TeamSite editions are complete copies of the Website created during the course of its ongoing development and maintenance.

Site Rollback

Site Rollback allows instant recovery of any earlier version of the Website.

Access Control

TeamSite respects the native operating system file and directory permissions necessary to provide strict access control for enterprise-level security.

LDAP Support

Authentication to the TeamSite system can be controlled via lightweight directory access protocol (LDAP) v3 integrating TeamSite with corporate security policy.

10.8. Portal

The Portal component is implemented with Viador Portal Suite COTS product. It is used by SFA application developers to integrate and distribute information and services in a secure, Web-enabled environment. The product provides the following portal functionality:

- Stringent authentication process for intranet and extranet users
- Pass-through of authentication if it is performed elsewhere
- Links to favorite Websites and other content sites
- Repository for storing portal information and documents
- Channels / Publish / Subscribe model to facilitate distribution of information
- User-customizable GUI
- Repository to host metadata information about files stored on the physical disk system of the Web server
- Portlets to support registration and integration of third party applications

Viador provides the SFA enterprise information portal framework. This framework delivers a range of services and supports the development and integration of portlets. Portlets are component based approach to enterprise information integration.

10.8.1. Single Point Of Access

The Viador Portal provides a single point of access for all corporate data. It is designed to collect information from a wide variety of heterogeneous data sources into a single repository. Users can search the repository, or the portal can be configured to alert users when new, pertinent information is added. Viador also delivers access to unstructured data

such as Web and documents, spreadsheets, engineering diagrams, and so forth. In addition, the Portal framework delivers access to corporate data sources, decision support databases, Internet news content, and enterprise applications.

10.8.2. Corporate Customization

Companies can customize the portal to provide access to a range of applications and data sources to users. Roles can be defined by the system administrator that can be applied to categories of users, such as marketing, sales, engineering, manufacturing, accounting / finance, human resources, customer support, or senior managers. Corporate customization also includes defining security policies and procedures, and defining how data will be categorized in the portal.

10.8.3. End User Personalization

Users can personalize their portal interface by selecting from the catalog of data sources and applications they have access to according to their assigned roles. The personalization process allows users to quickly locate information they require and filter out information that is extraneous. This increases productivity by reducing the amount of time wasted locating and consolidating information situated in a variety of heterogeneous sources.

10.8.4. Viador Portal Architecture

The following diagram illustrates the Viador Portal Architecture.

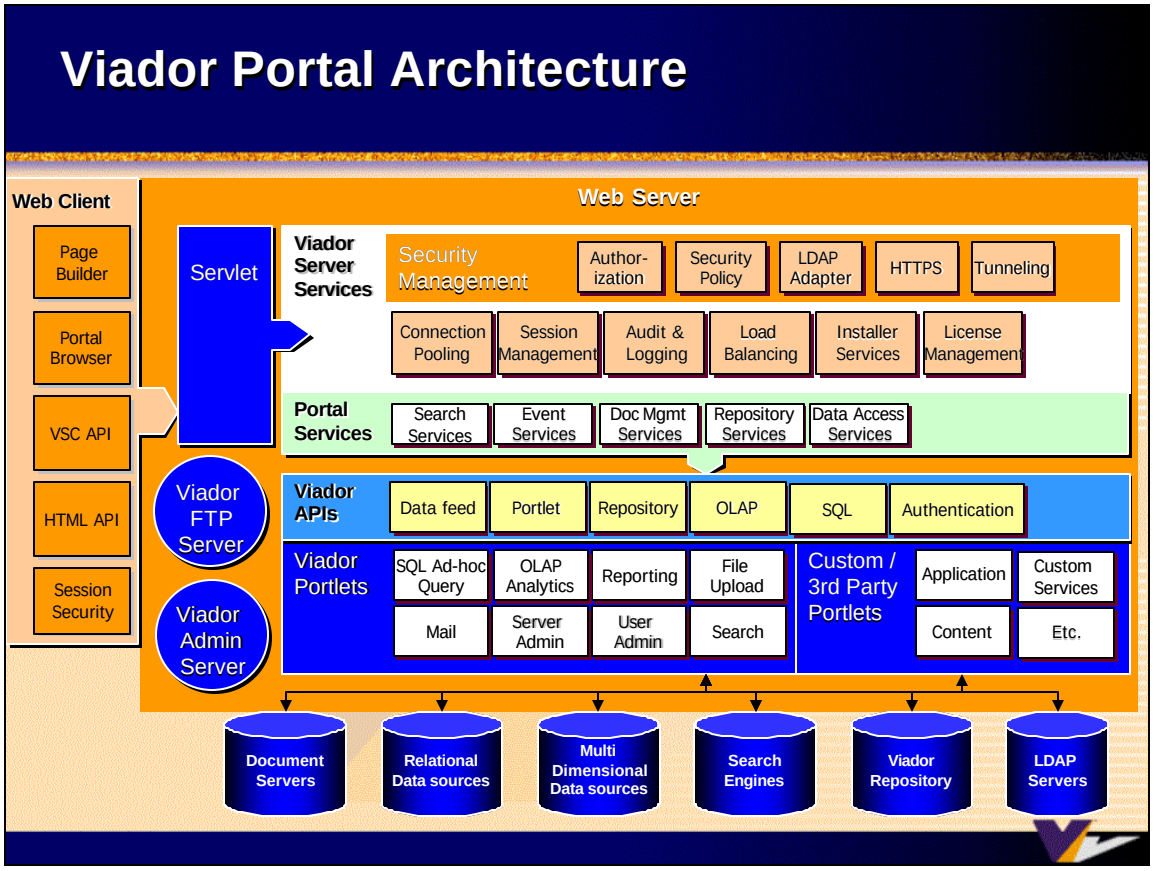

Figure 32 - Viador Portal Architecture

10.8.5. Viador Key Components

Viador Information Center (VIC) Server

The VIC provides the portal implementation. It is implemented in Java and is normally installed on the Web Server. The VIC interacts with Web Servers through Java Servlets.

Viador Repository

The Viador Repository is implemented as a RDBMS. The RDBMS can be Oracle, Sybase, DB2 or SQLServer. The SFA implementation is Oracle8i. Users can upload and register files of any type.

Viador Gateways

Viador Gateways provide an interface to COTS products. The Viador Universal Metadata Adapters (UMA) provide metadata translation for other COTS products.

Viador Servlets

The Viador Servlets provide the interface from Web Servers to the VIC. The servlets manage stateful connections to the server.

Viador Services

The Viador Session Manager provides session handles for client sessions and manages session timeouts.

10.8.6. Viador Technical Architecture

The following diagram illustrates the Viador Technical Architecture.

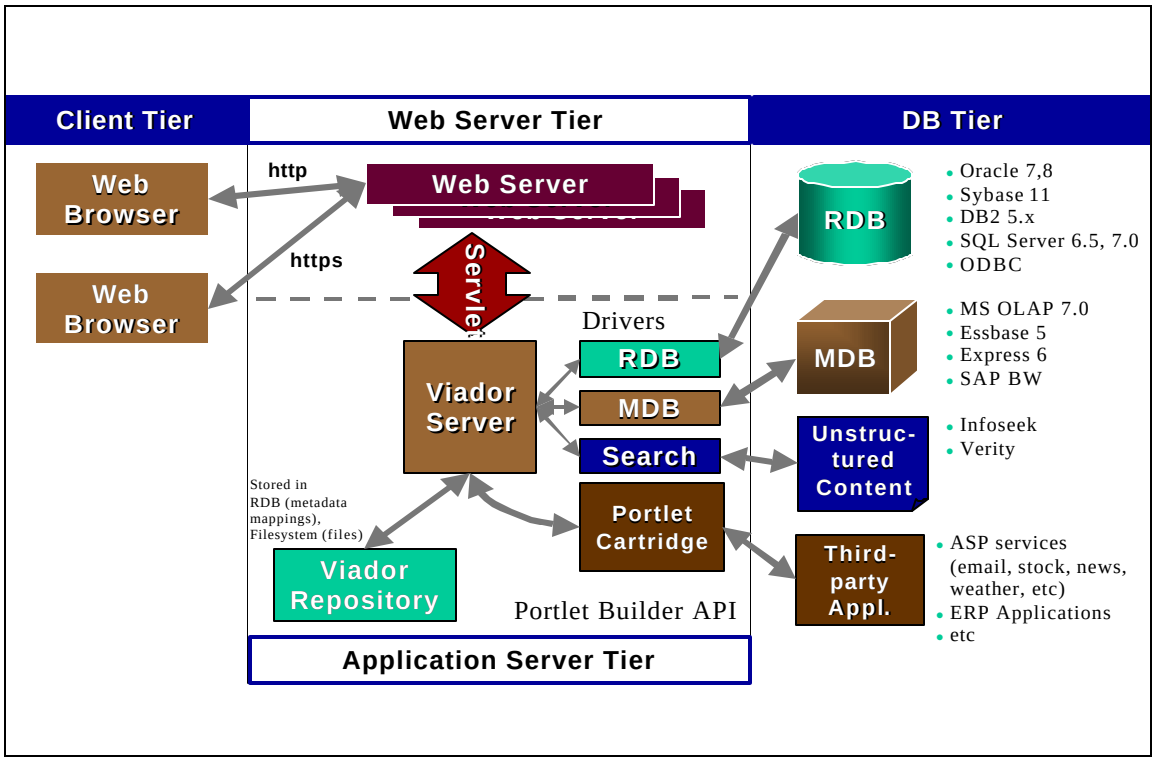

Figure 33 - Viador Technical Architecture

10.8.7. Viador Portlets

Viador Portal introduces the concept of portlets. Portlets are content or application services that are registered with the Viador Information Center and can be controlled and displayed by any Viador portal interface.

There are several types:

- Content portlets—e.g. Web pages and content feeds
- Application portlets—e.g. calendars, instant messaging, conferencing
- Portal tools—e.g. Viador business intelligence applications such as the Report Designer, OLAP Designer, the Portal Explorer

The following diagram illustrates Viador Portlets for Interwoven, Autonomy, and the MicroStrategy COTS products.

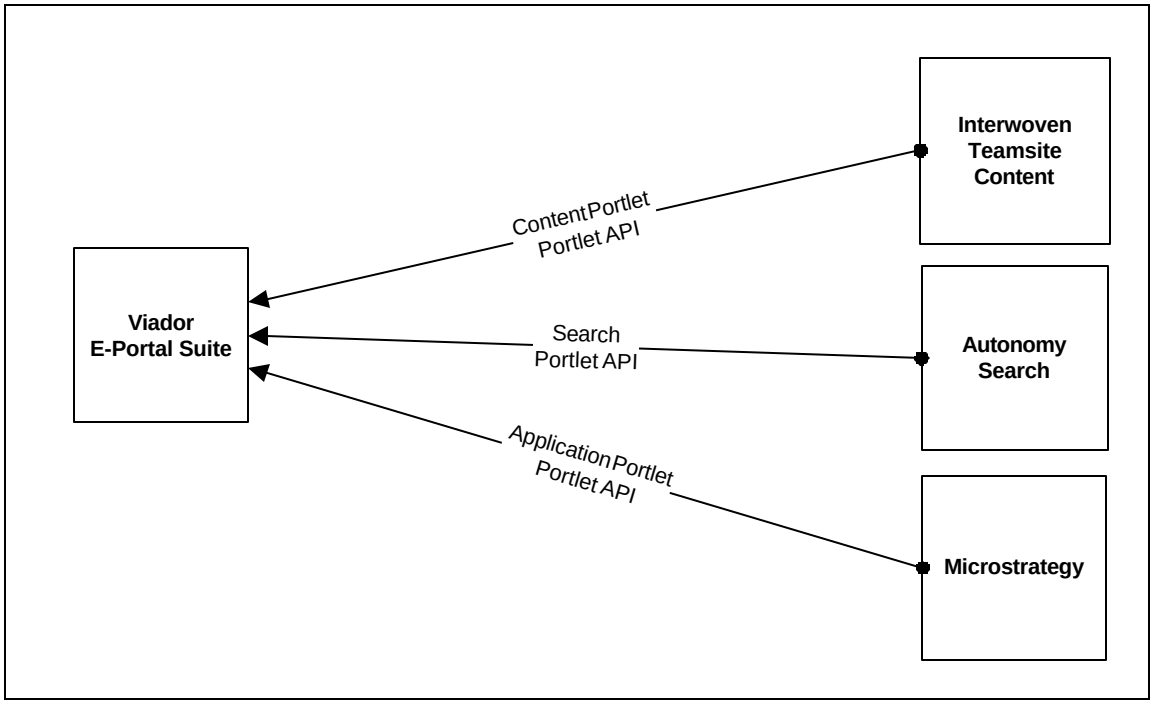

Figure 34 – Viador SFA Portlets

Portlet integration points include:

- A direct or parameterized URL link to a Web page
- A reference to a Java object that is run within the Viador Application Server

Portlet API

A developer can write a Viador portlet to provide special information to the user. The portlet can provide any added function, which the developer can write.

The Viador Portal launches the portlet. The portlet is a Java object that implements the Viador interface PortletIfc. When the Portlet has been launched, it returns an array of bytes to the Viador Portal, which in turn sends the bytes to the Web Browser component. Those bytes could be anything, which a Web Browser component can display; most commonly HTML for a Web page. The preferred method to invoke a portlet is to send an HTTP GET or POST request to the Viador servlet. This request passes the session ID, and specifies which portlet should be launched.

The HTTP Get or Post can specify the portlet in a few ways:

- The request might give the fully-qualified name of the Portlet class
- The request might specify an object which is associated with that classThe following diagram illustrates the Viador Portlet Architecture.

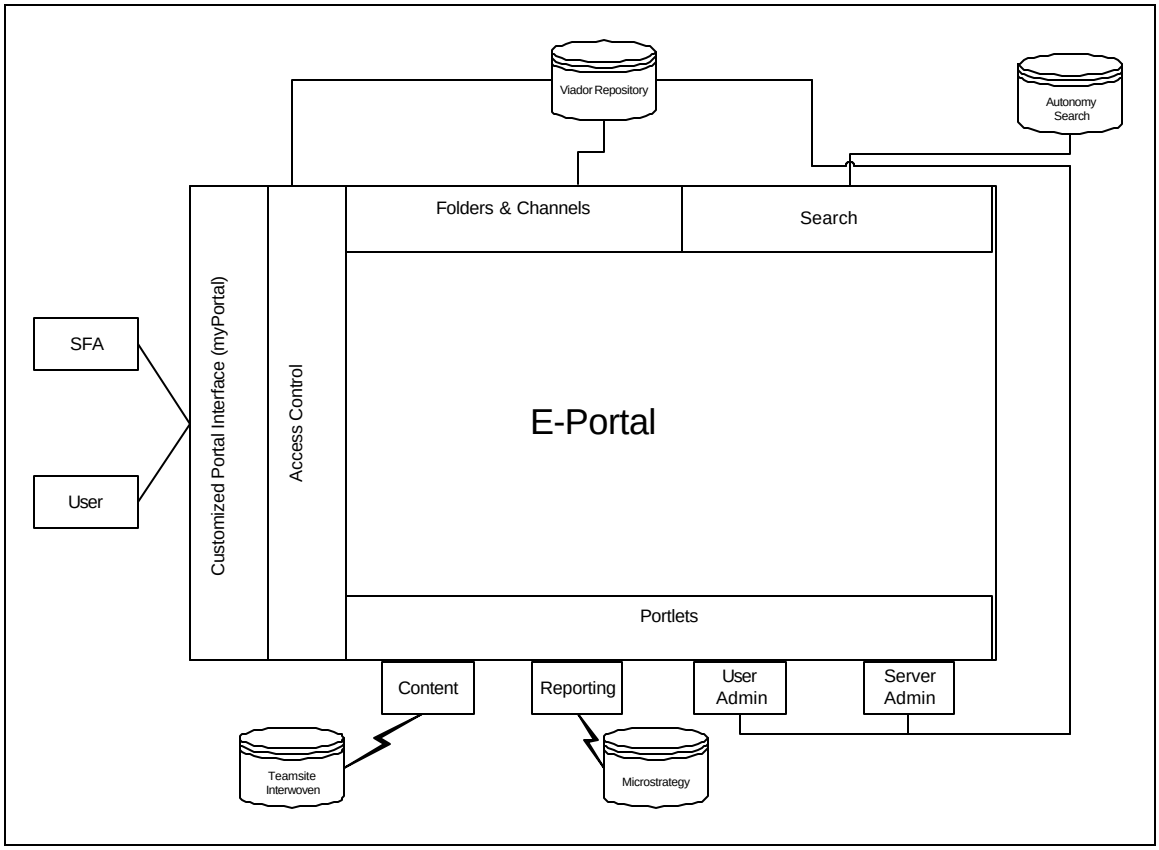

Figure 35 – Viador Porlet Architecture

The following diagram illustrates the Viador Control Flow.

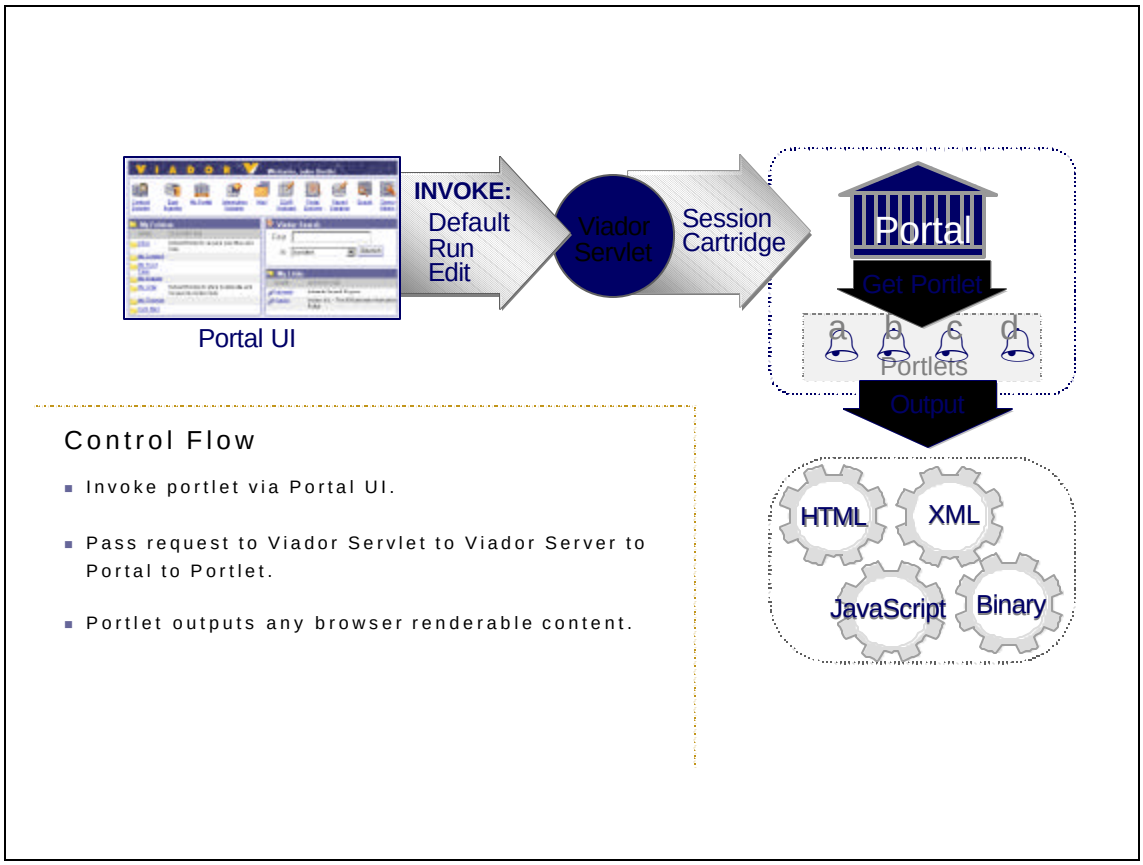

Figure 36 – Viador Control Flow

The portlet is usually launched by a user Portal window. For example, if the user double-clicks on a SQL Report, the portal window sends a request to the Viador Servlet, asking it to open the appropriate object. The Viador Servlet then checks the Viador Repository to find out which portlet can open that kind of object; it launches the appropriate portlet, and passes it the object.

A portlet must implement the Portlet API's PortletIfc interface. The interface specifies all the methods that the Viador Portal might need to call.

The PortletIfc interface requires four methods:

- Init
- HandleService
- GetURL
- NotifyEvent

Init

The Portal calls the portlet's init method right after the portlet is created. The Portal passes in its own object identifier. Also, in order to use any of the repository objects, the portlet will need to know the object identifier of a Util object for that session. If the portlet will need access to the repository, locate a Util object by calling the Portal's getUtil method and save it to a class variable.

HandleService

The Portal invokes this method to instruct the portlet to perform some task; the method returns an array of bytes, containing an appropriate return value for that task. After the portlet has initialized itself, the Portal calls this object to tell the portlet to take one of three actions:

- Open some object (usually a file) for viewing
- Open some object (usually a file) for editing
- Launch, but not open any file (the default service)
- Any other service which the portlet may define (if requested through a GET or POST command to the Viador Servlet)

When the method is called for one of these services, it is expected to return data, which can be displayed in a Web page. The Portal serves this data to the client Web Browser component. The portlet may define any other services it likes. For example, a form on a client machine might submit information to the portlet by invoking a portlet-defined SUBMIT_DATA service. The portlet could then process the information, take any appropriate actions, and then send a new form or a confirmation screen back to the client browser. Any Web-enabled application can send an HTTP GET or POST command to the Viador servlet, instructing it to invoke the handleService method of whatever portlet it specifies. The Portal invokes the portlet's handleService method, and returns that methods return value as the result of the GET or POST command. This way, any Web page can send and receive data from the portlet, and through it, from the Viador repository.

GetURL

This method is passed by any Java object. The portlet may either return a URL that, somehow, displays the object, or it may return null, signalling that it doesn't know how to handle such an object. The usual behavior is to provide a URL that opens the Viador servlet and instructs it to open the object. The servlet then passes the request to the Portal object, which takes the appropriate action. If the object is a Viador file object, the Viador portal would find the corresponding file, determine which portlet can open it, and launch that portlet. The Portal object and the AbstractPortlet class both provide a getURL method that correctly handles most ordinary objects. Most portlets getURL methods can simply call the Portal getURL, and return the Portal returns. If a portlet wishes to return a different URL for any object, it is free to do so.

NotifyEvent

The Portal calls this method to notify the portlet whenever some important event occurs. If the user logs out from the Portal, or is disconnected because of prolonged inactivity, the Portal will call this method.

10.9. Knowledge Management

Autonomy's search function provides a technology infrastructure for the automatic exploitation of content. Using unique pattern matching technology that conceptually profiles content, the Autonomy infrastructure is able to derive an understanding of the context and meaning of information. User interaction patterns with individual pieces of information are used to build an understanding of the interests and skills of the users. Applications are then implemented to provide users with personally relevant, timely information, as well as allowing them the ability to contact other users with common interests.

Autonomy uses a combination of Bayesian Inference and Shannon's Information Theory, to automatically assess and extract the key conceptual aspects of any document, or piece of unstructured information. Bayesian Inference is a mathematical technique for modeling the significance of ideas based on how they occur in conjunction with other ideas. Shannon's Information Theory provides a mechanism to extract the most meaningful ideas in these documents.

10.9.1. Scope and Application

The Autonomy search engine will provide the ability to use various types of information searches, offer personalized profiling, and features to search both structured and unstructured data.

10.9.2. General Architecure

The Autonomy COTS product can be represented as modules layered around the fundamental technology. The following diagram illustrates the Autonomy Architecture.

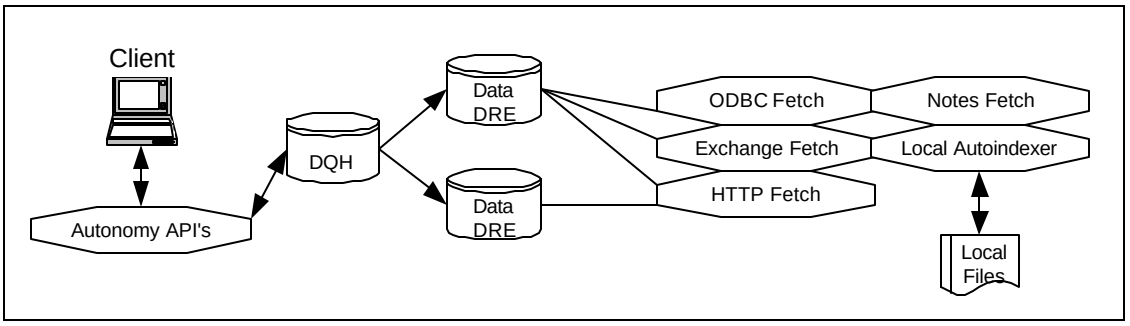

Figure 37 - Autonomy Architecture

10.9.3. User (Client) Workstation

The minimum workstation requirements to access information from the Autonomy search engine must include an Intel Pentium 266MHz CPU with 32MB of RAM. Local processing may come from accessing applications (i.e. MS Word, MS Excel, WordPerfect, etc....) that will be needed to access documents of the sort that have been indexed into the Autonomy database. The client application will need a network card and must be able to access the network via TCP / IP. The workstation must include Windows 95 or greater and a Web browser that is capable of running with the Windows environment (i.e. MS IE 4.01 or greater; Netscape 4.5 or higher).

10.9.4. Software Components

The Autonomy search capability is composed of three components. These components are the Dynamic Reason Engine (DRE), the HTTP Fetch Spider, and AutoIndexer.

10.9.5. Search Engine with the Dynamic Reason Engine (DRE)

The central component of the Autonomy search engine is the Dynamic Reasoning Engine (DRE). It is a scalable, multithreaded engine which performs the essential tasks of automatic import and conceptual profiling of content; delivery of results; and interaction with other components. The DRE can operate automatically to receive new content and perform the necessary tasks to exploit content and match it with user's interests.

Autonomy's Distributed Query Handler (DQH) allows for the interaction of multiple DRE, residing on the same or multiple physical servers, to transparently service a request. One DQH can be connect to up to 99 DREs. The DRE is able to import and deliver a wide variety of information types from disparate sources, as well as being able to import and deliver profiles of user interests. The DRE can also notify users about relevant information, and to also put users in contact with other users with similar interests.

The following diagram depicts the two examples of a DRE implementation, as well as how a single DRE can be a cluster of many DREs on one or more physical servers.

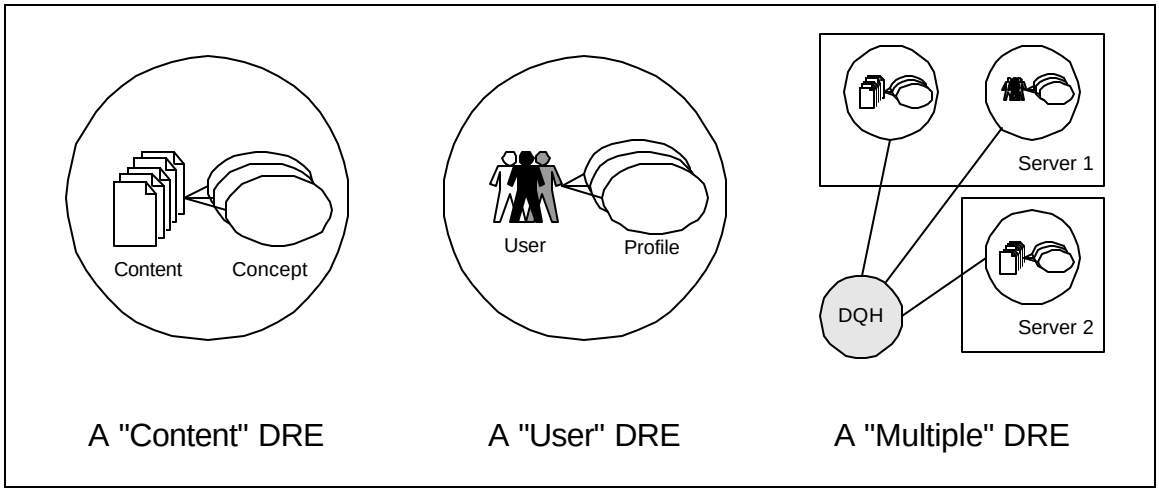

Figure 38 – Example DRE Implementations

10.9.6. HTTPFetch Spider

The HTTPFetch is a "spidering" feature that gathers documents from other Internet or Intranet sites and indexes them into an Autonomy DRE. The gathering steps (or fetch process) are spidering, importing and indexing. The first step is spidering. Spidering is the process of gathering information from a given site.

The HTTPFetch uses HTTP requests as a means to gather documents from remote Internet or Intranet sites. Once a document has been obtained, it is automatically analyzed for possible links to other documents. These links, if found, are followed conditionally based on the settings specified in the set-up configuration, and further documents are retrieved. The documents that were retrieved are then automatically imported into Autonomy's indexing file format and then indexed. Each instance of the HTTPFetch can "spider" multiple Websites simultaneously, making efficient use of processor and bandwidth resources.

10.9.7. AutoIndexer

AutoIndexer is another "spidering" feature that allows documents residing on shared network drives to be gathered and indexed into an Autonomy DRE.

AutoIndexer is a feature that polls directory structures on shared network drives for documents that have been created by external applications (i.e. MS Word, MS Excel, WordPerfect, etc...). The Autonomy search engine can support over 200 file formats. Once AutoIndexer has obtained a document from a directory it will automatically analyze the document and index certain information about the document into the Autonomy DRE. This information contains the link to the document in the directory, the summary of the document, the date the document was created and modified, and as an option the document content can be indexed into the Autonomy DRE.

10.9.8. Network Communication

Network communication with the Autonomy DRE is based on the TCP / IP sockets mechanism, in a distributed, client / server architecture. All Autonomy modules (such as the Autonomy HTTPFetch) communicate with the DRE via an IP address and an index port number. The index port number can be any unused system port number. The user query requests communicate with the DRE via an IP address and a query port number. The query port number can be any unused system port number.

10.9.9. Using Structured and Unstructured Data Searches

Structured data is stored in a relational database (Oracle, Access, DB2). Unstructured data includes all other types of data (e.g. documents, spreadsheets, and HTML files).

Autonomy can index both structured and unstructured information into the DRE. The indexing process uses different spiders depending on the data source that needs to be spidered. The Autonomy spiders that spider information from different data sources are listed in the table below.

Table 46 – Autonomy Spider

Spider	Information
HTTPFetch	indexes documents from the Web
AutoIndexer	indexes documents from the file system
Lotus Notes Fetch	indexes e-mail and documents in a Lotus Notes Database
Exchange Fetch	indexes documents in a Exchange Public Folder repository
ODBC Fetch	indexes structured information in a ODBC repository
ODMA	Indexes documents from a EDMS repository that is ODMA compliant
POP3 Fetch	Indexes e-mail from a POP3 mail server

During the indexing process, key information about the document is stored and added to the DRE using the Autonomy patented algorithms. Such information includes a link to the document (located in the original data source repository), the title of the document, a summary of the document, a document create date, and the concepts of the document. Autonomy never copies the documents, it provides a link back to the original repository. This means Autonomy will never compromise the security of the document that has been set in the repository.

Indexing structured information is different than indexing unstructured information, however both types will be indexed into the same DRE. Once the spiders have indexed both types of information, users can access the indexed information. The Autonomy DRE is the central data repository to query all data sources that have been indexed through a common interface.

10.9.10. Structured Information

The Fetch Spider is used to index structured information and uses ODBC Fetch. The ODBC Fetch uses an ODBC data source to gather content from a database. These tables or views contain the content to be indexed into a DRE. Other tables include the configuration, template, main and secondary tables. The configuration table defines how the tables relate to one another, the template tables specify how the data is stored. Each fetch has a main table where each row in this table represents a single document in the DRE. The secondary tables contain supplemental information for each document. Each row on the main table is read from the database and indexed into the DRE.

During the fetch process, the individual documents are given a unique reference id that specifies the row of the table where it was derived. The IP Address and location of the ODBC Content Retrieving CGI prefix this reference id. When a document, originating from an ODBC database, is returned as a link in Knowledge Update or Knowledge Server, clicking on the link will automatically pass control to the ODBC Content Retrieving CGI, which returns the results to the user in its original format. If the fetch (which retrieved the information) was configured to store the content in the DRE, then the content may be returned as normal plain text, with the loss of its original formatting. This method decreases the time required to return the content.

10.10. Directory Server

The initial release will not include Directory Services. However, it is possible that subsequent releases might use Directory Services for security services.

The Directory Services component is used to supply authentication and authorization information for the security services. The authentication process validates access to the Application Server component, and it can also authenticate for access to the Database Server component. Such information would include the location, capabilities and various attributes (including userid / password pairs and certificates) of resources and users, known to the ITA.

The initial release will use Netscape LDAP and DNS in lieu of Directory Services.

10.11. File Storage

10.11.1. Introduction

This section of the document provides an overview of the IBM AFS Enterprise File Systems that is used in the SFA project for the Department of Education. AFS is part of the IBM WebSphere Performance Pack Version 2. AFS provides the SFA system with the file sharing capability.

10.11.2. How AFS Works

The name space concept starts with a typical desktop client computer. All AFS client machines run a Cache Manager process. The Cache Manager maintains information about

the identities of the users logged into the machine, finds and requests data on their behalf, and keeps retrieved files on local disk. The local caching reduces network traffic and server load.

The central AFS file name space is key. While all the file and directory names in the name space are related to physical data on the servers, the names are visible in a single, enterprise-wide, and consistent directory structure. The servers perform all their work in concert to provide a seamless file name space visible by all desktops and all users. Any changes to the name space, any new files or any changed data are publicized to clients immediately.

Unlike past systems, which made use of the /etc/ file system on a client to map between a local directory name and a remote file system through a mounting operation, AFS does its mapping (file name to location) at the server. This process has the advantage of making the served file space location independent. Location independence means that a user does not need to know which File Server holds the file. The user only needs to know the path name of a file.

AFS, the File Sharing component of IBM WebSphere Performance Pack, has important applications in HTTP environments.

Typically an HTTP daemon sends to Web browsers the files available from the machine on which it executes. The process does not care whether the files that it serves are stored on a local disk or on an AFS file server since access to AFS files is indistinguishable from access to local files on an AFS client machine.

Many HTTP servers will run with files on the local server, so when you see a path, it is mapping to a file that is stored locally. From the HTTP server's perspective, there is nothing different from pulling a file out of the AFS directory structure. The Cache Manager and all the other elements that go into providing this unified name space make it possible for the HTTP Server to make use of the AFS client to access a file from the AFS directory tree.

10.11.3. AFS Concept

AFS is a distributed file system that allows users to share and access all of the files stored in a network of computers as easily as they access the files stored on their local machines. The file system is called distributed because files may reside on many different machines, but are available to users on every machine.

AFS stores files on a subset of the machines in a network called File Server machines. File Server machines provide file storage and delivery service, along with other specialized services, to the other subset of machines in the network, the client machines. These machines are called *clients* because they make use of the servers' services while doing their own work. In a standard AFS configuration, clients provide computational power, access to the AFS files and other general-purpose tools to the users seated at their consoles. There are generally many more client workstations than File Server machines.

AFS servers run a number of *server processe*s, so-called because each provides a distinct specialized service: one handles file requests, another tracks physical location of data, a third manages security, and so on. To avoid confusion, AFS documentation always refers to *server machines* and *server processe*s, not simply to *server*s.

AFS Cell

An AFS *cell* is a collection of client and server machines that form an administrative unit. A cell includes the machines, the AFS data on the File Server machines, the AFS databases on the Database Server machines and the accounts created to access the collection of machines and their resources. The cell name is the second element in the fully qualified path of AFS data. For example, data in the transarc.com cell would reside under /afs/transarc.com in the directory tree.

Each AFS user account and each AFS machine belongs to exactly one cell. This is referred to as the user's or machine's local cell. A user may be able to access other cells, which are referred to as *foreign* cells.

Single File Name Space

Although AFS cell is administratively independent, it is recommended to organize the local collection of files (*file space* or *directory tree*) in a way so those users from other cells can also access this information. AFS allows cells to combine their local file spaces into a global file space, and does so in such a way that file access is transparent. Users do not need to know anything about a file's physical location in order to access the file. All they need to know is the pathname of the file, which includes the cell name. Thus every user at every machine sees the collection of files in the same way, meaning that AFS provides a uniform name space to its users.

AFS Volumes

AFS groups files into volumes, making it possible to distribute files across many machines and yet maintain a uniform name space. A volume function like a container for a set of related files, keeping them all together on one partition. It is a collection of related files and directories and is part of the directory tree. This enables server administrators to break the directory tree down into smaller units that can then be stored on File Servers. Volumes can vary in size, but are by definition smaller than a partition, since partitions hold one or more volumes. It is recommended that maximum volume size to be 2 GB. You can read or write to

AFS volumes that are larger than 2 GB, but you cannot perform typical AFS volume operations, such as dumping, restoring, moving or replicating the volume.

Volumes are important to server administrators and users for several reasons.

Their small size makes them easy to move from one partition to another, or even between machines. The server administrator can maintain maximum efficiency by moving volumes to keep the load balanced evenly. In addition, volumes correspond to directories in the file

space; most cells store the contents of each user home directory in a separate volume. Thus the complete contents of the directory move together when the volume moves, making it easy for AFS to keep track of where a file is at a certain time. Volume moves are recorded automatically, so users do not have to keep track of file locations.

Replication and Caching

AFS incorporates special features on server machines and client machines that help make it efficient and reliable.

Replication

On server machines, AFS allows administrators to replicate frequently accessed but infrequently changed volumes, such as those containing Web pages with product information. Replication also called a replica, means putting an identical read-only copy of a volume on more than one File Server machine. The crash of one File Server machine housing the volume does not interrupt users' work, because the volume's contents are still available from other machines. Replication also means that one machine does not become overburdened with requests for files from a popular volume.

Caching

On client machines, AFS uses caching to improve efficiency. When a user on client workstation requests a file, a process named Cache Manager, running on the client sends a request for the data to the proper File Server machine. The user does not need to know which machine this is; the Cache Manager determines file location automatically. The Cache Manager receives the file from the file server and puts it into the cache, an area of the client machine's local disk or memory dedicated to temporary file storage. Caching improves efficiency because the client does not need to send a request across the network every time the user wants the same file. Network traffic is minimized, and subsequent access to the file is especially fast because the file is stored locally.

Cached Data Validation with Callback

When AFS client machine, requests a file from the AFS File Server machine, the Cache Manager first checks whether that file is already present in the local cache in the AFS client machine. This is true if that file has already been requested. If the file is not there, then the Cache Manager requests it from the AFS File Server Machine, gets the file, stores it in its local cache and serves the file to the application that has requested it (for example, a text editor).

The AFS File Server machine keeps a list of all the AFS client machines that have requested and cached data associated with a certain file in their AFS cache (either RAM or disk). When that particular file is changed by a user or process that is entitled to do so, the AFS File Server machine directly contacts all the AFS Clients that requested that file in the past, informing them that newer data is now available. This process is also known as callback.

The AFS client's kernel can safely return cached data to the application until it is notified by callback. When callback is received, the client must retrieve a fresh copy of the data since the cache is out of date.

If an AFS client receives callback from the AFS server, it marks the cached data as invalid. However, to reduce useless network traffic, the AFS client will retrieve a fresh copy of the file only when the client really needs it. This particular distributed approach places a lot of responsibility on the client but speeds up the user's response. Most of the data is read from the local disk unless the original file is changed. Network traffic is reduced since local caching requires fewer file accesses and new copies of cached files be requested only if needed. We also want to point out that last save wins. This means that if two users have opened the same file, the first user saves changes, then the second user saves the changes as well, then the second user overwrites changes made by the first user. AFS is not a revision control system.

Content Backup

The user's directory, the work area, is called home volume. A snapshot of the user's directory is usually made every night. Based on the design of the AFS, this snapshot is a volume. It is called a *backup volum*e, which can also be attached to the file name space to allow easy retrieval of files mistakenly deleted.

It is common for backup volumes to be created for all volumes in the system, not only user volumes, so the archive process can use this stable and consistent image when writing the data to tape. The server administrator can set up the system, so users themselves can correct careless file deletions with no system intervention. Notice that the creation of a backup volume every night is not automatic. The server administrator sets this up if desired with a cron job, called bos cron job, using the Basic OverSeer (BOS) Server, This is similar to a UNIX cron job in that it runs at a specified time, but under control of the BOS Server.

10.11.4. AFS Security

Even in a cell where file sharing is especially frequent and widespread, it is not desirable that every user has equal access to every file. One way AFS provides adequate security is by requiring that servers and clients prove their identities to one another before they exchange information. This procedure, called mutual authentication, requires that both server and client demonstrate knowledge of shared secret information (such as a password) known only to the two of them.

Mutual authentication guarantees that servers provide information only to authorized clients and that clients receive information only from legitimate servers. AFS utilizes algorithms and other procedures based on Kerberos. The Massachusetts Institute of Technology's Project Athena originally developed this technology.

Users themselves control another aspect of AFS security, by restricting access to the directories they own. Every AFS directory has an ACL. For any directory a user owns, the user can build an ACL that grants or denies access to the contents of the directory. An ACL pairs specific users (or groups of users) with specific types of access rights. There are seven

separate access rights and up to 20 different users or groups of people may appear on an ACL.

10.11.5. AFS Architecture

AFS is a distributed file system that uses the client / server model of computing. AFS servers are physically secure machines that store AFS files and directories, databases of AFS information, and AFS configuration files. Server administrators and users to access information from AFS servers and issue commands to interact with AFS servers use AFS client machines (for example, create protection groups or manage server processes).

AFS Servers

The processes it runs determine an AFS server's responsibilities. The responsibilities of AFS servers include:

- File Server - A machine running AFS File Server processes stores files. It also delivers files to AFS clients and receives files from AFS clients.
- AFS Database Server - A machine running AFS Database Server processes stores and maintains. Its responsibility include the following:
  - The Authentication Database stores an encrypted password for each account, an encrypted server key, and account security information.
  - The VLDB stores information about volumes and their physical location.
  - The Protection Database contains information about AFS users and the corresponding user IDs, AFS groups and their corresponding group Ids and group membership.
  - The Backup Database contains information used by the AFS Backup System to dump data to tape (for example, data that will be dumped together, the schedules that will be followed, and contents of tapes).

AFS System Control Machine (SCM)

A machine functioning as an AFS System Control Machine (SCM) stores master copies of AFS system configuration files and distributes these files to other AFS server machines that request them. It is important that the configuration files are the same on each AFS server in a single environment. They contain information about the machine's local cell, the names of Database Servers in the cell, server encryption keys, and a list of server administrators. One SCM per cell must be defined.

AFS Binary Distribution Machine

A machine functioning as an AFS Binary Distribution Machine (BDM) stores master copies of AFS binaries for a particular operating system and distributes these files to other AFS server machines that request them. One BDM may be defined for each operating system type. In this configuration, the AFS binaries can reside on one machine and have all AFS servers of the same operating system type pull the binaries from that machine. It is not necessary to

have a BDM, although it is recommended if you have multiple servers of the same operating system type.

AFS Clients

An AFS Client machine runs the Cache Manager, a set of kernel modifications responsible for communicating with processes on AFS servers and accessing AFS files and directories. The Cache Manager is responsible for communicating with processes on AFS server machines and accessing AFS files and directories.

The AFS File System

AFS files and directories are organized into a directory tree. The root of the directory tree is /afs. All AFS client machines see the same directory structure under /afs. This is known as the single file name space.

To access an AFS file or directory, the user just specifies the path. The user does not need to know the physical location of the file or directory. In fact, the physical location could change and the user would continue to use the same path. This is known as location independence.

The diagram below depicts a sample AFS file system.

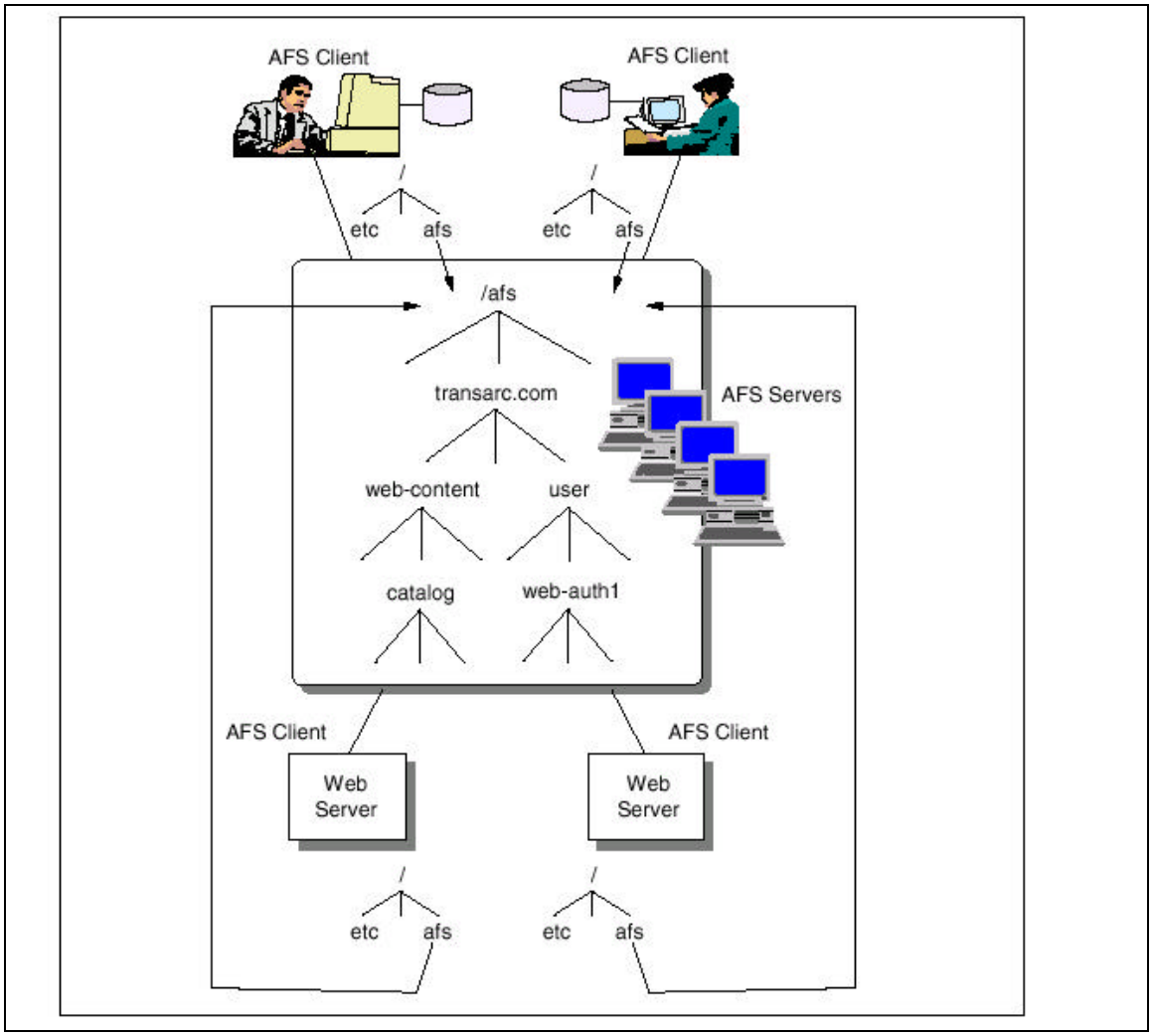

Figure 39 - AFS File System

Volumes

The AFS directory tree is broken down into smaller subtrees. Each subtree is a collection of related files and directories called a volume. AFS volumes provide a unit of administration for replicating frequently accessed files and directories, moving data from one physical location to another. AFS volume size should be restricted to 2 GB. It is possible to have a volume larger than 2 GB, but then it will not be possible to perform certain administrative tasks, such as dumping, restoring, moving or replicating the volume.

The diagram below depicts the AFS volume structure.

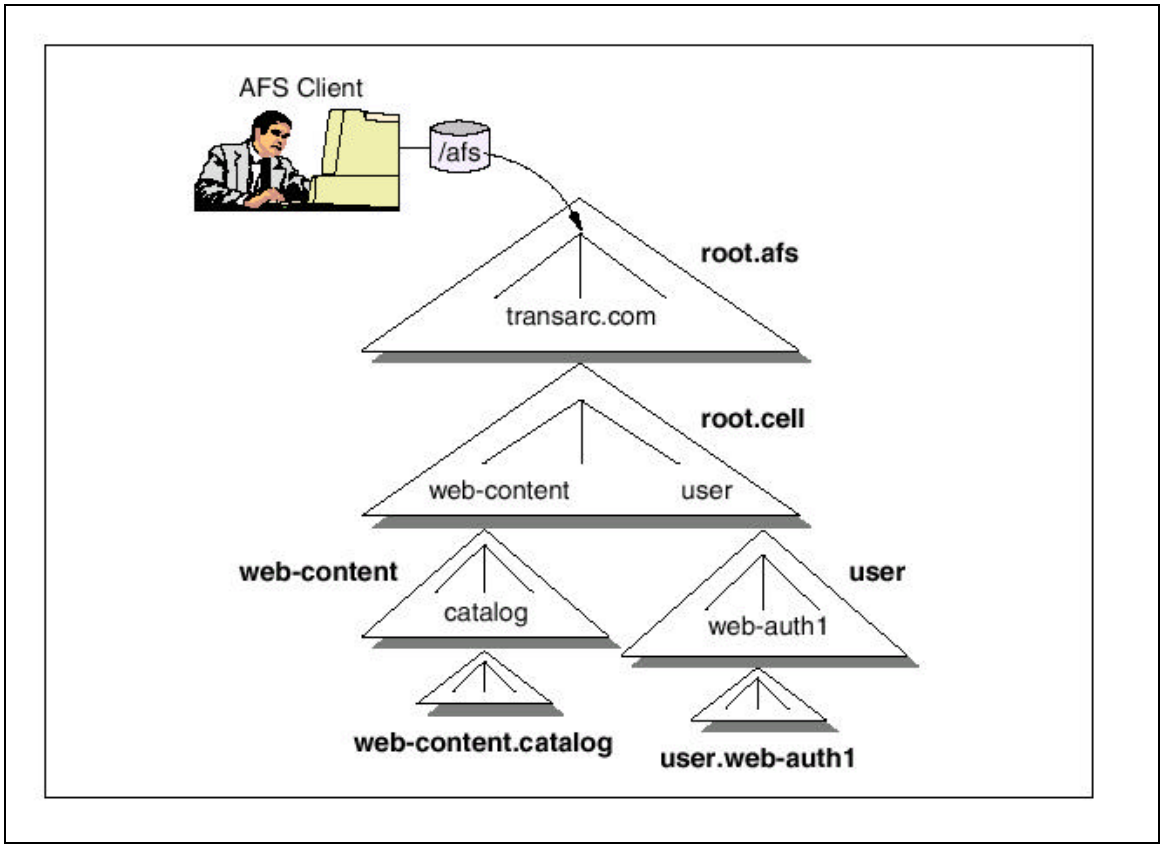

Figure 40 - AFS Volume Structure

Each AFS volume is identified by a name. The root volume is called root.afs by convention. The next volume is typically the root.cell volume.

AFS volumes are stored on the disk partitions of File Server machines. AFS partitions are often referred to as *vice partitions* because they have names in the form /vice*p*x, where *x* is a letter in the range from a to z and from aato iv. AFS partition names must be unique on a per server basis. In other words, it is possible to have a partition called /vicepa on the server wspp1 and also have a partition called /vicepa on the AFS server wspp2. The AFS partition size is constrained by the underlying operating system.

Mount Points

The point at which a volume is attached to the AFS directory tree is called a *mount poin*t. To users, mount points look like subdirectories. A mount point contains the name of the volume being attached to the directory tree. It does not contain information about the volume's physical location.

The diagram below depicts the AFS mount points.

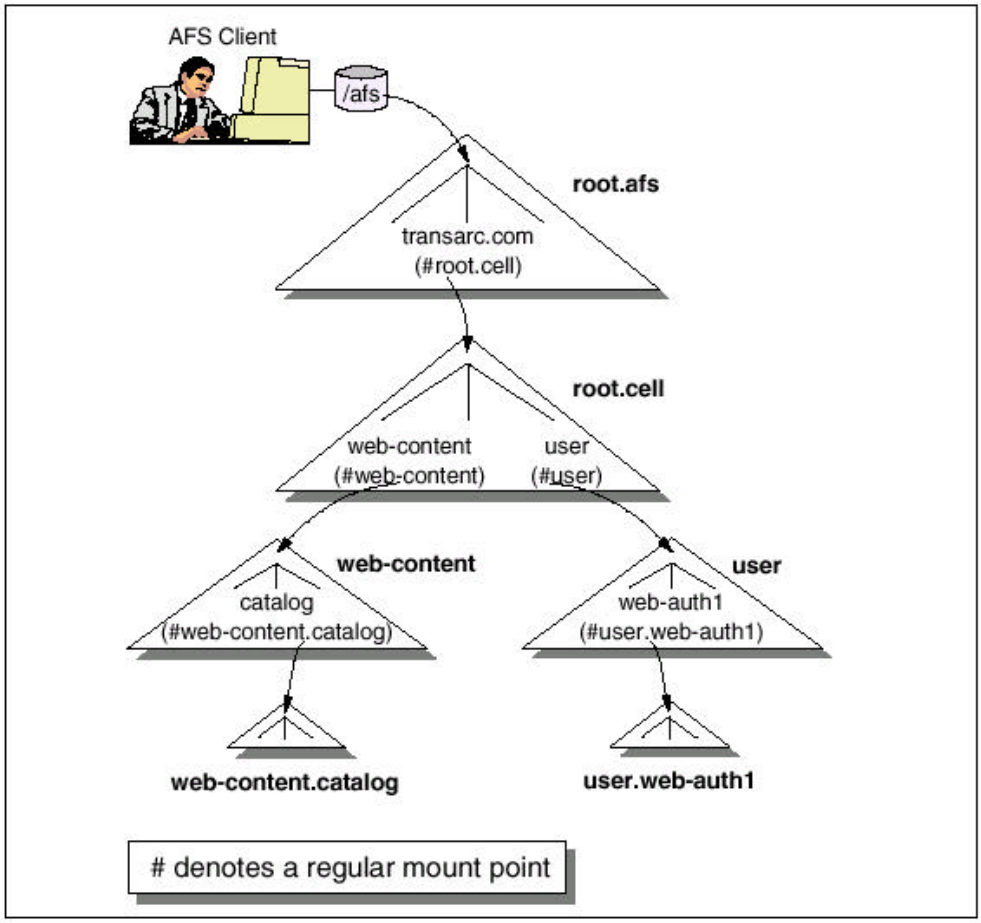

Figure 41 - AFS Mount Points

Volume Location Database (VLDB)

When a user specifies a path for an AFS file, the AFS client machine's Cache Manager interprets the path. The Cache Manager can extract the volume name from the mount point. To access the volume, the Cache Manager must determine where the volume resides, so it contacts the VLDB. The VLDB notifies the Cache Manager of the volume's location(s). The Cache Manager then contacts a File Server on which the volume resides and requests the file.

Caching

When the Cache Manager receives the requested file, it caches the file on the AFS client machine's local disk. Subsequent requests for the file may use the cached copy instead of fetching another copy from the File Server.

Callback Mechanism

When the File Server sends the file to the Cache Manager, it also sends a callback. The File Server will notify the Cache Manager, by breaking the callback, if the file changes. When a user requests a file that is already cached, the Cache Manager checks to see if the file has a valid callback. If so, the request is satisfied from the cache. If the callback has been broken, the Cache Manager retrieves the file from the File Server.

Protecting AFS Files and Directories

ACLs are used to protect AFS data. Every AFS directory has an ACL that controls access to the directory and its contents. The ACL is applied at the directory level, not the file level as in UNIX. An ACL consists of entries. Each entry specifies who has permission to access the directory and files within the directory. The entry also specifies what access rights are granted to that user or group of users. There are seven access rights, or permissions, that you can assign on an ACL:

- Lookup (l) - This permission allows you to list the contents of the directory and examine the directory's ACL. It is necessary in conjunction with other permissions
- Read (r) - This permission allows you to read the contents of files in the directory
- Insert(i) - This permission allows you to add new files or subdirectories to the directory. You can do this by creating the files with an editor or by copying the files from another location
- Write (w) - This permission allows you to modify the contents of files and to use the UNIX chmod command on files in the directory
- Delete (d) - This permission allows you to remove or move files from the directory
- Lock(k) - This permission allows you to run programs that set an advisory lock on files in the directory
- Administer (a) - This permission allows you to change the ACL on the directory

Server administrators and users can define AFS protection groups that contain a list of AFS users or machines. This protection group can be added to ACLs to give all members of the group the same access rights.

AFS Commands

There are two types of AFS commands: simple commands and compound commands. Simple commands consist of one word. Syntax is available for each of these simple commands by issuing the command with the help switch. Compound commands consist of two components: a command suite and a subcommand.

AFS Authentication

Authentication is proving the user's identity to AFS. A user, prior of being authenticated by AFS, has access to public AFS data, data that the *system:any*user group has permission to access via ACLs in addition to the client machine's local disk that the user has permission to

access via the operating system's permissions. After the authentication, the user has additional access to protected AFS data, data that the user or the user AFS protection group that the user belongs to has permission to access via the ACLs.

Authenticating is accomplished by logging in via an AFS modified login or by issuing the klog command. AFS includes a modified version of login for most AFS client machine supported system types. AFS integrated logon for Solaris involves installation and configuration of a Pluggable Authentication Module (PAM). This module integrates authentication mechanisms, such as login, to provide security infrastructure for all authenticated access to and from the machine. This modified login looks like the local operating system login, but the user is authenticated to AFS at login time. At login time, the user specifies a user name and password. If the password matches the user's password in the Authentication Database, the user is logged in to the local operating system and is granted an AFS token. If the password does not match the user's password in the Authentication Database, the user does not receive a token, but the user is logged in to the operating system if the password matches that of the local operating system.

To successfully authenticate via AFS modified login, AFS user names must match those of the local operating system. On Windows NT, passwords must also match. A token is evidence that the user is authenticated in a cell. Tokens have default lifetime of 25 hours, although server administrators can change this. The maximum token lifetime is 720 hours (30 days). When a token's lifetime is exceeded, the token expires and the user must re-authenticate. A user can re-authenticate before a token expires to refresh the token's lifetime.

AFS Server Concepts

AFS servers are physically secure machines that store AFS files and directories, databases of AFS information, and AFS configuration files.

Types of AFS Servers

There are four types of AFS servers:

- AFS File Server
- AFS Database Server
- AFS System Control Machine
- AFS Binary Distribution Machine

An AFS server can, and usually does, have multiple responsibilities. For example, it is common to configure an AFS server to function as a File Server and a Database Server. You do this by running the processes that give the machine these responsibilities.

AFS Server Processes

All AFS servers run the BOS Server process, bosserver. This process is responsible for starting and monitoring other AFS server processes on that machine. It knows which processes to

start based on configuration files the server administrator creates on the local machine. A machine functioning as an AFS File Server runs the fs trio of processes:

- The fileserver process runs continuously during normal operation to handle requests at the file and directory level. This is the process you interact with when you access data from AFS file space.
- The volserver process also runs continuously during normal operation and handles operations at the volume level. This is the process you interact with when you create, delete, or move volumes.
- The *salvager* process checks the system for internal consistency, and repairs errors it finds. It is analogous to UNIX fsck, but unlike fsck, it understands AFS volume structures and only operates on AFS data. The salvager runs automatically at startup time if the bosserver determines that the server was not properly shut down. It also runs automatically if the fileserver process fails. A server administrator can invoke the salvager automatically via the bos command suite, via the salvager command, or via the AFS Control Center's Server Manager.

While these are three separate processes, the BOS Server treats them as a unit and understands dependencies between the processes. A machine functioning as an AFS Database Server runs separate processes for each of the four AFS databases:

The kaserver process is responsible for maintaining the Authentication Database. It also verifies user identity at login by requiring a password, grants the user a token as proof to AFS server processes that the user has authenticated, and provides a way for:

- server and client processes to prove their identities to each other. The kas command suite is used to interact with the kaserver process
- The ptserver process is responsible for maintaining the Protection Database and providing information when the fileserver process needs to retrieve information about an AFS user or protection group. You can use the pts command suite to interact with the ptserver process
- The vlserver process, not to be confused with the volserver process that runs on File Servers, is responsible for maintaining the VLDB and for providing the Cache Manager with information about volumes and volume location. The vos command suite is used to interact with the vlserver process
- The buserver process is responsible for maintaining the Backup Database and for providing an interface to the AFS Backup System. The backup command suite is used to interact with the buserver process
- The Update Server processes are responsible for transferring information from SCMs and BDMs to other AFS servers. These processes include upserver and upclient. Although the name may suggest otherwise, these processes run on AFS servers, not AFS clients:
  - The upserver process runs on SCMs and BDMs and is responsible for distributing the contents of a specified directory, such as /usr/afs/etc on an SCM or /usr/afs/bin on a BDM, to other servers that request them.

- The upclient process is responsible for checking a specified directory on a particular machine and retrieving any files that have changed.

In addition to these processes, AFS servers may also run the runntp process. This process synchronizes an AFS server's clock with the clock on another machine. It is important for the clocks on all AFS servers in a cell to be synchronized for AFS services to function properly. If there is another time-keeping mechanism already running on the server, it is not necessary to run this process.

Local Directories

AFS creates and relies upon information in a number of directories on a server's local disk:

On UNIX systems, the /usr/afs/etc directory contains cell-wide server configuration files that must be the same on every AFS server in your cell. The bos command suite is used to update the files. The files include:

- CellServDB - The CellServDB file is an ASCII file that contains a list of the local cell's Database Servers
- UserList - The UserList is an ASCII file that contains the names of system administrators in the local cell
- KeyFile - The KeyFile contains server encryption keys. A server encryption key is a string of octal numbers that is used to encrypt/decrypt packets of information. This file contains one key that matches the key in the Authentication Database for the AFS entry
- ThisCell - The ThisCell file is an ASCII file that specifies the cell to which the server machine belongs

On the UNIX platform, the /usr/afs/local directory contains machine-specific configuration files. These files differ from server machine to server machine. The commands from the bos command suite are used to update the files. The files include:

- BosConfig - The BosConfig file contains information about AFS processes that the bosserver is responsible for. It defines which processes should be running on the server and the state they should be in. This file also stores information that specifies when the bosserver checks for new binaries in its /usr/afs/bin directory and when it restarts all of the machine's AFS server processes.
- SALVAGE.fs - The SALVAGE.fs file controls the behavior of the Salvager. If this file exists when the bosserver starts, the bosserver will run the Salvager before starting other File Server processes. This file is created when the bosserver starts the fileserver process and is removed when the bosserver properly shuts down the fileserver process.
- salvage.lock - The salvage.lock file is a zero-length file that exists when the Salvager is running. The purpose of this file is to allow only one server administrator to run the Salvager on the File Server at a time.
- NoAuth - The NoAuth file is a file that should not be present under normal circumstances. If present, AFS server processes are not required to check authorization.

- Sysid - The sysid file contains up to the maximum number of IP addresses supported for each server machine; this number in AFS 3.4a and 3.5 is 15. The sysid file also contains a unique host identifier which distinguishes the server from all others in the cell. The file is automatically created at startup time. Since this file contains server-specific information, server administrators should not copy this file from one File Server to another.

On UNIX systems, the /usr/afs/bin directory contains AFS server and command suite binaries. Since the binaries will be the same on every AFS server of the same operating system version, a master copy can be maintained on the BDM and the upclient/upserver processes are used to distribute the binaries to other AFS servers with the same operating system version.

If a UNIX machine is functioning as an AFS Database Server, the /usr/afs/db directory contains the four AFS database files, which are:

- kaserver.DB0, the Authentication Database
- prdb.DB0, the Protection Database
- vldb.DB0, the Volume Location Database
- bdb.DB0, the Backup Database

The /usr/afs/logs directory on UNIX systems contains log files associated with various AFS server processes. It may also contain old versions of the log files if they exist and core files generated by the AFS server processes. The log files found in this directory include:

- BosLog, generated by the bosserver process
- FileLog, generated by the fileserver process
- VolserLog, generated by the volserver process
- SalvageLog, generated by the salvager
- AuthLog, generated by the kaserver process
- VLLog, generated by the vlserver process
- BackupLog, generated by the buserver process

AFS Client Concepts

Server administrators and users to access information from AFS servers and issue commands to interact with AFS servers use AFS client machines (for example, create protection groups or manage server processes).

Client Responsibilities

Recall that AFS clients run the Cache Manager, a set of kernel modifications responsible for communicating with processes on AFS servers and accessing AFS files and directories. When a user requests an AFS file, the Cache Manager interprets the path to extract the volume name from the mount point, requests volume location information from the VLDB, fetches the file from the File Server, and caches the file locally.

Processes

The afsd process runs on AFS client machines. This process is started from the client machine's initialization file. The afsd process initializes the Cache Manager by transferring information into kernel memory and starting several AFS daemons. When the afsd process starts, some of its responsibilities include:

Setting a field in kernel memory that defines the cell to which the machine belongs

- Putting names and IP addresses of Database Servers from local and foreign cells into kernel memory
- Defining the directory under which the AFS directory tree resides, typically /afs
- Determining whether to use disk or memory cache
- Defining the name of the local directory to be used for caching (on UNIX, typically /usr/vice/cache)
- Setting cache size and cache parameters
- Randomly selecting a File Server in the local cell to get time from, unless the -nosettime switch is used with the afsd binary

Local Directories

On the UNIX platform, AFS relies upon information in two directories on a client machine's local disk:

The files necessary to configure the client machine are by default located in the /usr/vice/etc directory. They include:

- afsd binary file – This binary initializes the Cache Manager.
- cacheinfo file - The cacheinfo file stores information about the cache. It specifies the directory where the Cache Manager mounts AFS, the name of the cache directory, and the cache size.
- ThisCell file - The ThisCell file contains the name of the client machine's local cell.
- CellServDB file - The CellServDB file contains the names and IP addresses of Database Servers in the local cell and in foreign cells.
- modload (Solaris) subdirectory - The modload directory on Solaris contains files necessary to build a kernel that incorporates AFS modifications if they exist for the specific operating system.

The /usr/vice/cache directory contains files that the Cache Manager creates or maintains for its own use. The contents include:

- V- files - V- files are used to store chunks of data in the cache. The file names are in the form *Vdigits*, such as V3367. The number of V- files in the directory depends upon the size of the cache.
- CacheItems - The CacheItems file is a cache overhead file that stores information about each V- file in the cache.

- VolumeItems - The VolumeItems is another cache overhead file that records information about volumes that the Cache Manager has accessed. This includes volume-to-mount point relationships and volume-location information.

The @sys Variable

AFS uses the @sys variable to simplify administration of operating system-specific files. The @sys variable expands to the current system's CPU and operating system type, for example: sun4x_56 for Solaris 2.6. When the Cache Manager encounters the @sys variable in a path, @sys is replaced with the machine's system type. The following figure offers a graphical representation of the usage of the @sys variable on the Solaris platforms.

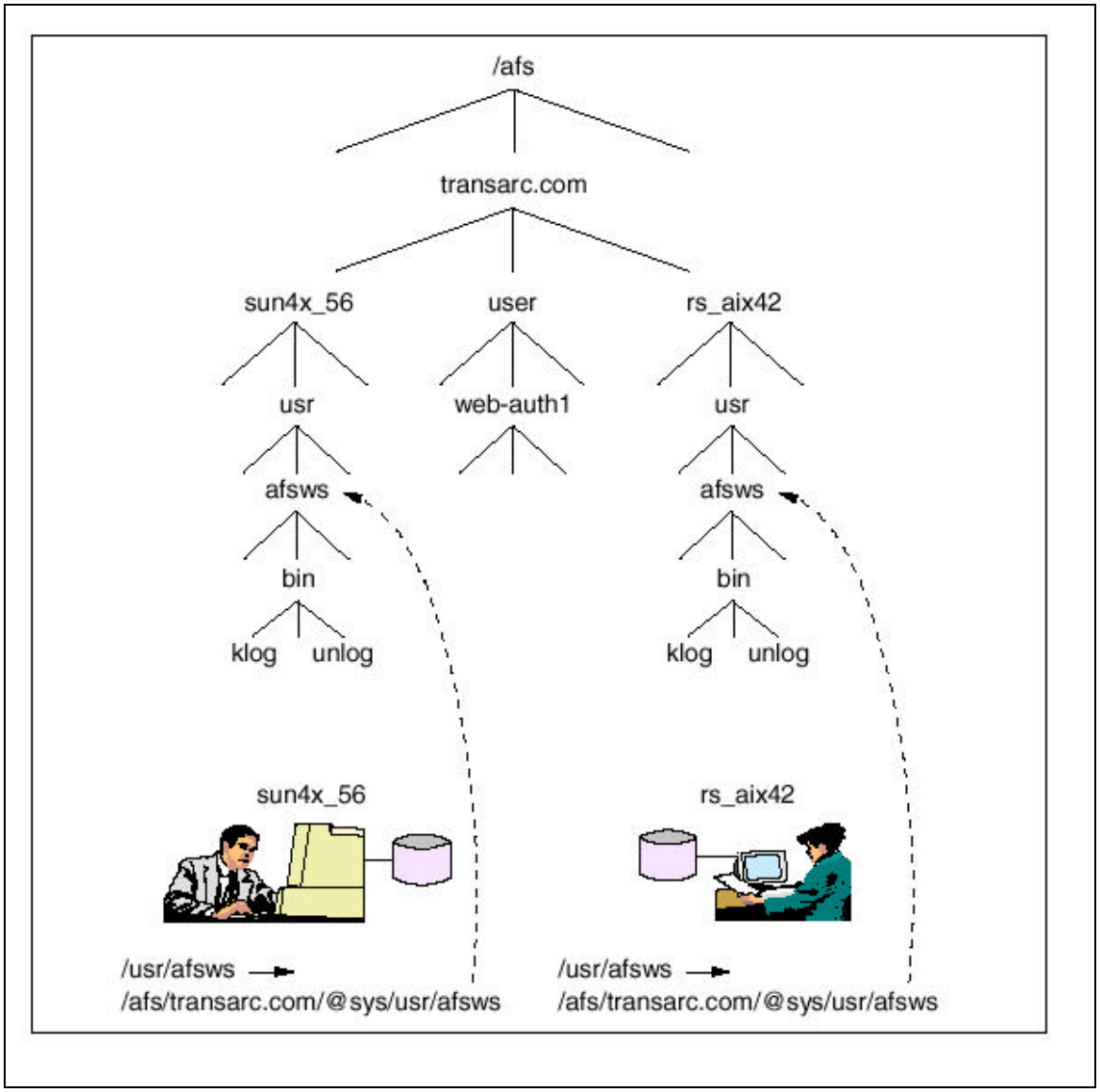

Figure 42 - @sys variable on the Solaris platforms

AFS Volume Concepts

The AFS directory tree is broken down into smaller subtrees. Each subtree is a collection of related files and directories called a *volum*e. AFS volumes provide a unit of administration for replicating frequently accessed files and directories, moving data from one physical location to another.

Volume Headers

Volume headers contain pointers to where the volume's files and directories are located on disk. Although an AFS partition is formatted as the vendor-supplied file system, data is stored in a different manner. If a server administrator logs in to an AFS File Server and lists the contents of a vice partition, the contents would include

volume headers. The following diagram depicts the relationship between the volume pointers, headers, files and directories.

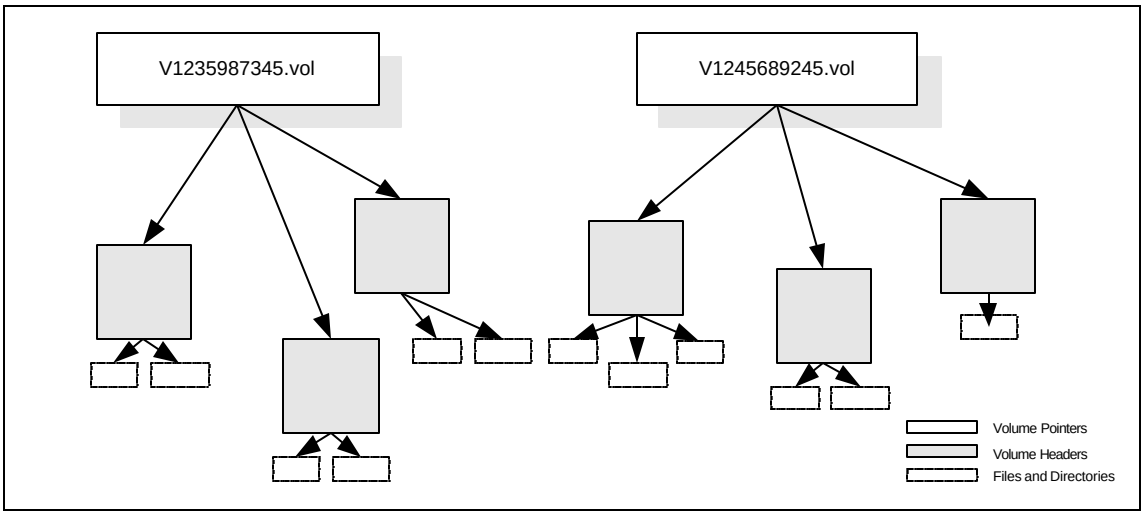

Figure 43 - Relationship Between the Volume Pointers, Headers, Files and Directories

Types of Volumes

AFS has three types of volumes:

- A ReadWrite volume contains modifiable copies of AFS files and directories. Each volume has a ReadWrite version.
- A ReadOnly volume contains a non-modifiable copy, replica, of the contents of a ReadWrite volume. Each ReadWrite volume can have up to eleven ReadOnly replicas. The name of the ReadOnly volume is the name of the ReadWrite volume with a .readonly extension (for example, root.afs.readonly).
- A Backup volume contains a non-modifiable snapshot of the contents of the ReadWrite volume. Each ReadWrite volume can have at most one Backup version. The name of the Backup volume is the name of the ReadWrite volume with a .backup extension (for example, root.afs.backup).

Volume Traversal Rules

The VLDB contains volume location information. A Cache Manager contacts the VLDB to learn information about a particular volume. The VLDB tells the Cache Manager where the volume and its replicas reside. If replicated, server preferences determine which server to contact for a replica. The decision whether to access the ReadWrite version or a ReadOnly replica is based on traversal rules.

Traversal rules define how a client chooses between the ReadWrite version and ReadOnly replica if the volume is replicated. The rules are as follows:

- If root.afs is replicated, use a ReadOnly volume.
- If root.afs is not replicated, use the ReadWrite volume.

As the path is traversed:

- If you are currently in a ReadOnly volume and the next volume is:
  - Replicated, you must use a ReadOnly volume
  - Not replicated, you must use the ReadWrite volume
- If you are currently in a ReadWrite volume, you must access a ReadWrite volume

Traversal rules assume that you did not force a mount point to contain the name of a ReadOnly volume. The following figure offers a graphical representation of traversal rules.

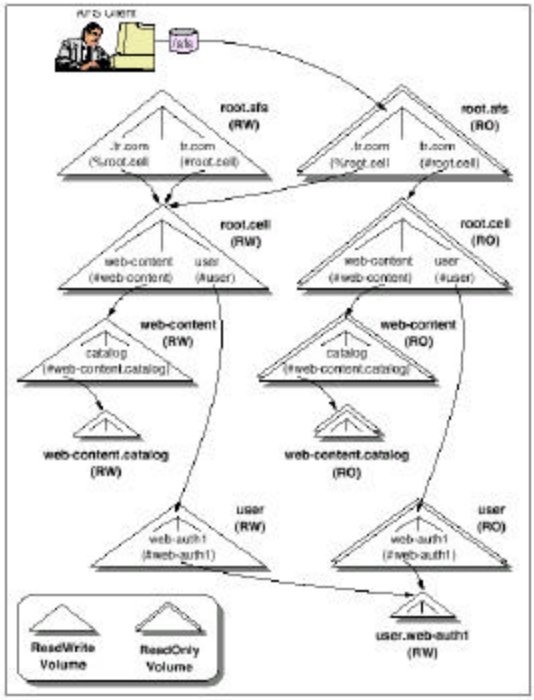

Figure 44 - Graphical Representation of Traversal Rules

In the root.afs volume, there is a ReadWrite mount point, .tr.com, that contains the name of the root.cell volume. We know this is a ReadWrite mount point because it contains a percent symbol (%) instead of a hash symbol (#).

The Cache Manager does not support the previously stated traversal rules when it crosses or interprets a ReadWrite mount point. Instead, the fact that a mount point is ReadWrite indicates that the Cache Manager should access a ReadWrite version of the specified volume.

By convention, if the Regular mount point that contains the name of the root.cell volume is called *cellnam*e, and the ReadWrite mount point that contains the version of the root.cell volume is called *.cellnam*e.

Since it is best to let the Cache Manager follow traversal rules, and ReadWrite mount points do not follow traversal rules, there should only be one ReadWrite mount point in your directory tree. In fact, when the Cache Manager is on a ReadWrite volume, it always accesses the ReadWrite version of the next volume.

Therefore, you only need one ReadWrite mount point high up in your namespace to put the Cache Manager on the ReadWrite path, and guarantee that the Cache Manager can access the ReadWrite version of all volumes.

When a Web author wants to access a file in the AFS directory tree, he specifies the path. Suppose he wanted to access /afs/tr.com/Web-content/filex. This would take the Web author to the file in the ReadOnly version of the Web-content volume. If the Web author wanted to access a modifiable version of the file, he must specify the path /afs/.tr.com/Web-content/filex because it leads to the ReadWrite version of the Web-content volume.

When a server administrator is setting up the AFS directory structure, it is important to create the ReadWrite mount point to the root.cell volume before replicating the root.afs volume. If the server administrator forgets to do this, it may not be possible to access the ReadWrite volumes in the directory tree and create mount points to attach volumes into the directory tree.

In particular, if a server administrator forgets to replicate the root.afs volume, replicated copies of all volumes mounted beneath root.afs will never be accessed. Suppose a server administrator forgot to replicate the root.afs volume but did replicate the root.cell volume in the tr.com cell. The replicas of the root.cell volume would not be used. There could be up to 11 replicas of the root.cell volume that are taking up space on the File Servers but are not being accessed. Auser would specify path /afs/tr.com to access the root.cell volume. In this case, the Cache Manager would use traversal rules to determine which version of the root.afs volume to access. Since root.afs is not replicated, the Cache Manager must access the ReadWrite version. The Cache Manager extracts the name of the root.cell volume from the tr.com mount point and contacts the VLDB to determine where the volume resides. Even though there are replicas of this volume, the Cache Manager uses traversal rules. The rules indicate that if you are in a ReadWrite volume, you must access a ReadWrite volume.

A similar situation occurs if you forget to replicate any volume in the directory tree. If you forget to replicate a volume, replicas of all volumes mounted beneath that volume would never be accessed.

Naming Suggestions

When you create a volume, you create its ReadWrite version. The name you select should be no more than 22 characters. The limit is really 31 characters, but includes the ReadOnly or Backup extensions. When you create a ReadOnly replica or a Backup volume, the ReadOnly and Backup extensions are automatically appended to the volume name.

The name you select for a volume should reflect the volume contents. Server administrators should be able to examine volume names from the VLDB or from volume headers and have an idea of what the volume contains.

The volume name should also reflect where the volume is attached in the directory tree. This is useful if you have to delete a volume. When you delete a volume, you should also delete the mount point that attaches the volume to the directory tree. If your volume name and mount point do not correspond, you will have to find another way to determine where the volume was attached to the directory tree so you can delete its mount point.

10.12. Database Server

None

11 Acronyms List

Table 47 – List of Acronyms

Acronym	Description
ACL	Access Control List
AFS	Andrew File System
AIX	Advanced Interactive Executive
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BDM	Binary Distribution Machine
BOS	Basic OverSeer
CAD	Computer-Aided Drawing
CAM	Computer-Aided Manufacturing / Methods
CBR	Content Based Routing
CDS	Cell Directory Service
CGI	Common Gateway Interface
CICS	Customer Information Control System
CMP	Container-Managed Persistence
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-the-Shelf
CPU	Central Processing Unit
DAS	Database Auto-Synchronization
DB	Database
DCE	Distributed Computing Environment
DHTML	Dynamic Hypertext Markup Language
DMZ	Demilitarized Zone
DNS	Domain Name Server
DOE	Department of Education
DQH	Distributed Query Handler
DRE	Dynamic Reason Engine
DTD	Document Type Definition

Acronym	Description
DTS	Distributed Time Service
EAI	Enterprise Application Integration
ESS	Enterprise Solution Structure
FDDI	Fiber Distributed Data Interface
FTP	File Transfer Protocol
GB	Gigabyte
GUI	Graphical User Interface
HP	Hewlett Packard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IA	Internet Architecture
IDL	Interface Definition Language
IIOP	RMI / Internet Inter-ORB Protocol
IMS	Information Management System
IP	Internet Protocol
IPSec	Secure Internet Protocol
IS	Internet Server
ISS	Interactive Session Support
IT	Information Technology
ITA	Integrated Technical Architecture
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNI	Java Network Interface
JSP	JavaServer Pages
JVM	Java Virtual Machine
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MAC	Media Access Control
MB	Megabyte
Mhz	Megahertz
MOFW	Managed Object Framework

Acronym	Description
ND	Network Dispatcher
NIC	Network Interface Card
NNTP	Network News Transport Protocol
NT	New Technology
ODBC	Open DataBase Connectivity
ODS	Operational Data Stores
OLAP	On-Line Analytical Processing
OMG	Object Management Group
PAM	Pluggable Authentication Module
PDF	Portable Document Format
POP3	Post Office Protocol 3
RAM	Random-access Memory
RDBMS	Relational Database Management System
RMI	Remote Method Invocation
RPC	Remote Procedure Call
SAN	Storage Area Network
SCM	System Control Machine
SFA	Student Financial Assistance
SMTP	Simple Mail Transfer Protocol
SQL	Structured Query Language
SQLJ	Structured Query Language for Java
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TP	Transaction Processing
UI	User Interface
UMA	Universal Metadata Adapters
UNIX	Universal Interactive Executive
URL	Uniform Resource Locator
VDC	Virtual Data Center
VIC	Viador Information Center
VLDB	Volume Location Database

Acronym	Description
VM	Virtual Machine
WAN	Wide Area Network
WAS EE	WebSphere Application Server Enterprise Edition
WAS	WebSphere Application Server
WLM	Workload Management
WTE	Web Traffic Express
XML	Extensible Markup Language

Appendix A: Detail of DRE Configuration Sections

[My Security] Section

The meanings of these sections together with the meaning of their key value pairs are covered in detail below. Please note that for settings that can have either of two values TRUE or FALSE, you can use any of the following values:

TRUE=ON=Y=1

FALSE=OFF=N=0

[License] Section

This section contains the licensing details, including the license holder's name and the license key. Do not edit this section, as this could stop the DRE functioning.

Table 48 – [Licencse] Section of DRE Configuration File

Key	Description
KEY	License key. E.g. KEY=**********
HOLDER	License holder's name. E.g. HOLDER=AUTONOMY

[Server] Section

This section governs the basic DRE operation.

Table 49 – [Server] Section of DRE Configuration File

Key	Description
QUERYPORT	The port number by which queries are sent to the DRE. E.g. QUERYPORT=7000
INDEXPORT	The port number by which indexing commands are sent to the DRE. If this line is not present or is set to 0, then indexing will not be available. E.g. INDEXPORT=7002
HYPHENS	When set to 1, the DRE will index words that are hyphenated as individual words and as one word. E.g. Off- line would have the words 'Off', 'Line' and 'Off-Line' indexed. When set to 0, the DRE will only index hyphenated words separately. E.g. Off-Line would be indexed as 'Off' and 'Line'

Key	Description
HYPHENCHARS	This indicates the symbol that represents the hyphen. E.g. HYPHENCHARS=- or / or * etc.
TERMSIZE	This represents a set number of characters per word to store. This is used when running different languages on the DRE. The default value is 10.
CHARCONV	Indicates the language in which the DRE is running. This can be either: European, Japanese, Korean, Thai, Simplified Chinese, or Traditional Chinese E.g. CHARCONV=2 For any language other than European please contact Autonomy for additional files.
STOPLIST=filename	Load stoplist file, list of words to be ignored from your search (words with 0 weighting). Defaults to STOPLIST2.DAT
STATUSDIR=path	Specify path for the DRE's Status Directory, directory which stores the temp files when indexing. Defaults to Status
QUERYCODE=code	Specify code string that must appear in query request
INDEXCODE=code	Specify code string that must appear in index request
QUERYCLIENTS/ INDEXCLIENTS	These are optional and do not need to be included in the DRE.ini. They are lists of IP Addresses (which can include wildcards), separated by commas, for machines permitted to send queries/indexer commands. If these lines are present then only machines with listed IP Address will have their index commands/queries serviced. E.g. QUERYCLIENTS=193.128.*,207* INDEXCLIENTS=* As Index Commands are potentially destructive, it is advisable to set INDEXCLIENTS=127.0.0.1 or INDEXCLIENTS=localhost. If these parameters do not have the S on the end the settings are ignored.
STRIPLANGUAGE	This too is optional and is used to select which language to use when stripping (e.g. running is stripped to run). The default setting is 0 options are: 0 for English, 1 is used for conversion from UK to US English, 2 is used for no stripping, 3 is for German. E.g. STRIPLANGUAGE=1
ECHO	When set to 1 and the DRE is run from an MS-DOS window rather than a service, then a DRE activity log is displayed in the DOS window. E.g. ECHO=1

Key	Description
LOGGING	When set to 1, this indicates that the DRE activity is logged to a file, rather than the active DOS window. E.g. LOGGING=1
LOGFILE	This is optional and indicates which file is to be used to log the DRE activity. The default is QUERYH.log E.g. LOGFILE=QUERYH.log
NOSUGGESTNUMS	If set to 1, this disables suggestions from terms which are numeric. Where you may specify a query it will ignore any numbers within the DRE. (See Section "Dynamic Reasoning Engine Commands") E.g. NOSUGGESTNUMS=1
QUERYSUMMARY	When set to 1, the DRE will return the first three lines of the document in a specified field named *Quick Summary* in the DRE.ini file. IMPORTANT: In the DRE.ini file, QUERYSUMMARY must be set to 1 and in the CFG file, located in the Scripts Directory, with the Autonomy Product name as its prefix, QUERYSUMMARY must be set to 'on'.
SUGGESTTERMS	This specifies the default number of terms that are used when performing a 'Suggest More' Command during Querying. (See Section "Dynamic Reasoning Engine Commands") E.g. SUGGESTTERMS=40 so the system takes the best 40 terms from the original document and uses them as a query which will produce further relevant documents.
COMBINE	When set to 1, this indicates that all sections of a document that were indexed separately are combined in the query result to show as one document. (See Section "Sectioning a document"). E.g. COMBINE=1
MAX_DOC_PER_STRCTDAT	This deals with memory allocation and should generally not be altered. E.g. MAX_DOC_PER_STRCTDAT=20000
PROPERNAMES	When set to 1, this indicates that a name consisting of more than one word must be combined and indexed as one word. E.g. Joe Bloggs would be indexed as one word if PROPERNAMES=1
MEMFIELDS	This is an advanced option and causes particular fields to be held in memory for faster structured field queries. The options for this parameter are: Enabled Disabled E.g. MEMFIELDS=1

Key	Description
FIELDSEARCH	When set to 1, all text contained in a fields will be indexed so they are searchable, use: FIELDNAME notation. Data will need re-indexing if this parameter is changed. The options for this parameter are: Disabled Enabled set as default
SOUNDEX=0 / 1	Index soundex words as well as normal words. Default 0
DEFAULT_XOPTIONS=string	String added to whatever xoptions are specified in a query. For options, refer to the xoptions options in the section "Query to DRE database(s)" of Appendix: "Low Level Commands to the Dynamic Reasoning Engine".
MAXQUERYWORDS=n	Max number of words allowed in a query. Default 200
MAXTOTALHITS=n	Max number of hits allowed in evaluating query. Default 1000000
MAX_QDOCS=n	Max number of docs to return from query, Default 5000
UNESCURL=0 / 1	Apply strUnescape (unescapes any variables that come to the DRE) to any cgi variables.
ROOT_STRCTDAT=path	Path to store STRCTn.DB files. Default . /
INDEXTITLES=0 / 1	Index titles as content. Default 1
MAXLOGSIZE=n	Max size of QUERYH.LOG. Default 1000000
INDEXNUMBERS=0 / 1	Enable indexing of numbers. Default 1
STORECONTENT=0 / 1	Stores the content in the DRE. Default 1
SEPARATORS=chars	Chars to use for separating the input text. E.g. SEPARATORS=, for a comma delimited file or a ' ' for a normal text file.
RECVTIMEOUT=n	Seconds after which a 'receive' times out. Default 10 seconds
MINFREESPACE=n	Minimum disk space required to allow indexing. DRE has internal minimum which varies according to index size.
SUGGESTBIAS=0 / 1	Increase weight of suggest results. Default 1
INDEXPRI=0 / 1	Give priority to indexer. If set to 0, queries will always have priority, but there is a possibility indexing will be unable to continue in the event of continuous sustained overlapping queries

[Schedule] Section

Schedule represents the procedure that the DRE performs to vacate the disk space allocation of where deleted records once were.

Table 50 – [Schedule] Section of DRE Configuration File

Key	Description
TIME	This specifies the time that the Compact Procedure will first be performed. *E.g. TIME=23:59*
INTERVAL	This specifies the number of hours that the DRE will wait before the next Compact Procedure will be performed. *E.g. INTERVAL=24*
COMPACT	This indicates the minimum number of deleted documents required for the Compact Procedure to take place. E.g. COMPACT=1
EXPIRE	This must be set to 1 if you want any of your databases to expire after the specified duration.

[Default]

This section governs the default configuration settings. The values in this section are employed for any job stated in the [Configuration] section, that does not have its own definition entry.

Table 51 – [Default] Section of DRE Configuration File

Key	Description
deleteEscapeReferences	Boolean value that when set to 1 will escape references when doing deletion. This is needed when trying to remove documents with spaces in the reference. You will need to use it with DRE version 3.0 and setting UNESCURL=1 in the DRE.INI
deletePathReplaceUpToSlash	If PollingAction=7 or PollingAction=8, this is used to specify a string that is to be replaced up to a certain'/'. This is used so that a single portion of many strings which contain different substrings can be replaced as opposed to just one particular word. E.g. In the case where the files in the queue are C:\a\b\hello,c:\a\c\hello And c:\b\a\hello deletePathReplaceUpToSlash=3 Would replace the portion up to the third back-slash in each string.
DeleteReferenceFromContent	Boolean value telling Autoindexer when it deletes files that the reference may have been obtained from the document content. The setting is either on (1) or off (0). Instead of using the usual file reference for example c:\data\myfile this setting enables the reference to be obtained from the document content.
DeleteReferenceStart	Start tag to be used when obtaining the reference from content. E.g. deleteReferenceStart=<DOCID>

Key	Description
DeleteReferenceEnd	End tag to be used when obtaining the reference from content. E.g. deleteReferenceStart=</DOCID> So using the deletereferencestart and end, the content found between the two will be used as the document reference. These settings only apply when deleting documents, as to delete all the Autoindexer does is send a delete command to the DRE with a reference. If you leave either of these start or end parameters blank then it defaults to beginning and end of file. You can also use \n, \r, \t in the settings so that you can delimit the reference by return characters.
directoryAfterDate	The number of days before today that the document must have been modified. E.g. directoryAfterDate=-1 (i.e. yesterday)
directoryBeforeDate	The number of days after today that the document must have been modified. E.g. directoryBeforeDate=5 (i.e. 5 days from today)
DirectoryCantHaveCSVs	This specifies the string that must not appear in the directory path of a spidered document. E.g. directoryCantHaveCSVs=*.sys,*.bat,*.exe
DirectoryFileMatch	This is a wild card specification of which files in the directory to process. E.g. directoryFileMatch=*
DirectoryMustHaveCSVs	This specifies the string that must appear in the directory path of a spidered document. E.g. directoryMustHaveCSVs=*/temp/*
DirectoryPathCSVs	This specifies the directory in which lie the files to be processed. E.g. directoryPathCSVs=D:\projects\autoindexer\files This can be a comma separated list of directories which means that more than one directory can be processed.
DirectoryPathRecurseMatchCSVs	This setting indicates a set of wildcards to match against whilst recursing the directory tree. This is different from directoryMustHaveCSVs in that the wildcard match is done against the recursion path not the full path of the file. Hence, it's more efficient as it won't recurse directories that it doesn't need to do. So for example: directoryPathCSVs=C:\files\ directoryPathRecurseMatchCSVs=*01*,*02* will process: c:\files\199901 c:\files\199902 It will not do: c:\files\morefiles\199901 c:\files\morefiles\199902 as the match is done at every recursive step.

Key	Description
DirectoryRecurse	This indicates whether or not to recurse into directories. It can be either: 1 or on for "Recurse" 0 or off for "Don't Recurse" E.g. directoryRecurse=off
DREHost	IP address where the DRE resides. Should you require to index into more that one DRE separate the IP addresses by a comma. E.g. DreHost=localhost,120.7.0.0
FilenameOutputMode	Boolean value where, if set to 1, will create a file listing all the files that are to be processed. It is used so that if you need to do file based processing you can create a list of files that need processing from a directory.
FilePollFilename	This specifies the filename from which the list of files to be processed is read. E.g. filePollFilename=queue
FileBaseDirectory	This is the directory path to attach to each file in filePollFilename. You can specify either the full paths or just the File names inside the queue file. E.g. BaseDirectory=c:\Files\
importIDXFilesAction	Specifies what to do with idx files that have been processed. This can be either: 0: delete IDX file 1: move the IDX file to another directory 2: leave in directory
importIDXFielsMoveTo	If importIDXFilesAction=1, this is the full path that specifies where the idx files are moved to.
ImportPathReplaceString	Specifies the string that will replace the substrings of files in the queue.
IndexOverSocket	This indicates if the file to be indexed is to be sent over the socket or if it is local, see IndexLocalFile
IndexPort	The port number by which indexing commands are sent to the DRE. If this line is not present or is set to 0, then indexing will not be available. If you are using more than one DRE specify the index ports separated by commas. E.g. INDEXPORT=2001,6001
MoveToDirectory	This specifies which directory to move to when PollingPostAction is set to 2. E.g. MoveToDirectory=D:\projects\autoindexer\processed\

Key	Description
PollingAction	This specifies the action to perform when polling. The value of this parameter can be either: 1 for 'indexing idx files to a DRE' (idx files are files that are in hash form. See Section 4.1) 2 for 'importing files in various formats and indexing them into a DRE' 7 for import+index+delete this will import files to an IDX format, index them and also delete any documents from the DRE that have been removed from the local file structure 8 'delete from a DRE' E.g. PollingAction=2
PollingPostAction	This specifies the action to perform after the file has been processed. The value of this parameter can be either: 0 for 'do nothing' 1 for 'delete the file after processing' 2 for 'move the file to another directory'. It will keep the subdirectory structure. The value of this key=value pair defaults to 0. E.g. PollingPostAction=0 When moving the originally scanned files somewhere else, the subdirectory structure is retained. BOOL FileCopyKeepDirStructure(charsRoot1, charsFilename, charsRoot2, charsMode)
PollingMaxNumber	This specifies the maximum number of files to be processed at each poll. E.g. PollingMaxNumber=100 The value of this key=value pair defaults to 100.
QueryPort	The port number by which queries are sent to the DRE. Again if you are using more than one DRE specify the query ports separated by commas. E.g. QUERYPORT=2000,6000
IndexLocalFile	Specifies if the local IDX file is to be indexed or if it will be sent over the socket. You can set either IndexOverSocket=1 or IndexLocalFile=0 (this is the default) or you can set IndexOverSocket=0 or IndexLocalFile=1. The two pairs are equivalent. If both settings are present then IndexOverSocket takes precedence.
IndexLocalFilePathReplace IndexLocalFilePathString	When you are doing IndexOverSocket=off or IndexLocalFile=on (both are equivalent) - that is you are sending the filename to the DRE rather than the data over the socket, you need to tell it how to change the path so that if the IDX file is in c:\hello\a.idx, but the DRE is on another machine, so for the DRE it will be d:\hello\a.idx then you can set these two parameters

Key	Description
IndexMode=REFERENCE	Allows the killing of duplicates using methods other than the standard URL / reference match. Options include: REFERENCE – reference ONLY MATCHNN – conceptual match ONLY (i.e. if the documents are similar above a certain threshold NN) REFERENCE2MATCHNN – This means that it will check all databases for reference duplicates rather than the one database that is being indexed into.

[Fields] Section

This section specifies the fields that are to be indexed in documents.

Table 52 – [Fields] Section of DRE Configuration File

Key	Description
NumFields=*n*	*n* is the number of fields to be included in the DRE.
Field0=DREReference Field1=Index10.DRETitle Field2=*myfield* Field*n*=FIELDNAME	This indicates the name of the *n*th field. Sequential fields to be indexed into the DRE. Field 0 and 1 are set as follows: Field0=DREReference. Field1=Index10.DRETitle Field2 onwards are user specified. The DRETitle should be specified with 'index10.' to indicate it is to be indexed, and have approximately 10% higher weighting. E.g. Field2=Summary
Fieldn=INDEX.Name Fieldn=MEM.sz,Name	INDEX.Name This means that when documents are indexed into the DRE, the field Name must be indexed as well, it is recommended that this be enabled. If Memfields=1 in the [Server] section, Fieldn ,it must have the following format: Fieldn=MEM.sz,name sz is the number of bytes to be stored in memory name is the name of the nth field The Mem value indicates that the field is to be held in memory for faster structured field queries. E.g. Field0=mem.224,summary= The full field is always saved to disk but only the first sz characters will be considered for the memfield query. Whenever the INDEX and MEM are used together, Index must come before MEM. E.g. Field0=INDEX.MEM.200,myfield Beware the MEM function will utilize computer resources, this facility is not necessary unless structured field searching is a constantly used process.

Key	Description
DATEFIELD=name	Informs the DRE that a particular field in the idx file contains the date. This field must come AFTER the DREDATE specifier in the idx file. For example, #DREFIELD mydate="1999/10/25" This field does not necessarily have to be declared as a structured field. You need to change DRE.INI thus: [Fields] DATEFIELD=mydate Field0=….etc Note the format: YYYY/MM/DD or Number of seconds since 1970 or NOW You can choose other names if required. In the event of there being multiple fields called DREDATE, the DRE will use the one that appears last in the idx file

[Cache] Section

This section is internal and the parameters contained should not be altered unless specified to do so by Autonomy.

[IndexSummary] Section

This section governs how the DRE indexes the summary of each document into a structured field at index time.

Table 53 – [Index Summary] Section of DRE Configuration File

Key	Description
METHOD=QUICK/CONCEPT	Set Summary Type: For METHOD=QUICK: MINWORDS=n — Set minimum length of summary MAXWORDS=n — Set maximum length of summary FIELDNUM=n — Set number of field to store summary in For METHOD=CONCEPT: FIELDNUM=n — Set number of field to store summary in SENTENCES=n — Number of sentences to include in summary.

[IndexCache] Section

This section specifies details of the cache used for indexing documents into the DRE. This is used to increase the indexing speed.

Table 54 – [Index Cache] Section of DRE Configuration File

Key	Description
MaxSize=n	Size of the index cache in Kb. The larger the value of MaxSize, the faster the indexing process. This is dependent upon the size of the individual idx files. The recommended maximum size is 20000. Past this size the potential to slow down the indexing process increases.

[dbname] section

This section is used to hold information about the database in the DRE with name *dbname*.

Table 55 – [dbname] Section of DRE Configuration File

Key	Description
ExpireTime=*nn*	*nn* is the number of hours that documents will reside in database dbname before they expire.
ExpireIntoDataBase= *name*	*name* is the name of the database in the DRE where expired documents from database *dbname* will be moved. If this is not specified then expired documents will be deleted from the DRE. Expiring databases from one database into another will only function if you store content.
MaxHits=*n*	*n* is the maximum number of documents to return from database dbname when the DRE is queried.
ReadOnly	Specifies that the database *dbname* is to be read only. Indexing will not be allowed.
StoreContent	This allows the database to store original content of documents
DateRange	This allows queries to be limited by date range for this database
Security*n*=*MySecuritySection*	*N* is the number of the database in the DRE that a security check corresponds to. The security check is used to only show relevant documents to a query that a certain user is allowed to see. *MySecuritySection* is the name of the section in the DRE.INI file that contains the settings for the security check corresponding to database *n*.

[MySecuritySection] Section

This section is used to hold information about security checks corresponding to certain databases. The security check is used to determine which documents certain users can access.

Table 56 – [MySecuritySection] Section of DRE Configuration File

Key	Description
Library=MySecurityDLL.dll	This specifies the DLL that performs the access control function determining which documents certain users can access.

Key	Description
Key=value	Multiple key=value pairs can be used E.g. ACLUsername=admin to set any additional parameters that the DLL needs to use. These settings are optional and will vary according to how the access control check is implemented. ACLPassword=adminpass

HTTPFetch configuration

This section details all of the HTTPFetch configuration parameters available, provides descriptions, default values, the range of possible values and specifies which sections the parameter can be used in.

[License] Section

This section contains the licensing details, including the license holder's name and the license key. Do not edit this section, as this could stop the POP3Fetch from functioning.

Table 57 – [License] Section of DRE Configuration File

Key	Description
KEY	License key. E.g. KEY=**********
HOLDER	License holder's name. E.g. HOLDER=AUTONOMY

[Default Section]

Table 58 – [Default] Section of DRE Configuration File

Key	Default	Range	Description
IndexMode	REFERENCE2MATCH	NA	Allows the killing of duplicates using methods other than the standard URL / reference match. Options include: REFERENCE – reference ONLY MATCHNN – conceptual match ONLY (i.e. if the documents are similar above a certain threshold NN) REFERENCE2MATCHNN – This means that it will check all databases for reference duplicates rather than the one database that is being indexed into.

Key	Default	Range	Description
Nsockets	16	1-512	The total number of sockets to share between the spiders
SpiderCycles	Infinite	0	Number of times to repeat (-1 for infinite
SpiderStartTime			The time the spider will start
SpiderRepeatSecs	100000	1-	Number of seconds between each cycle
SpiderRepeatInterval	Null		This expresses the time between each cycle in human readable form eg # hours
SpiderPath	./		Path to spider files
SpiderSetFlowRate	4096kbps for a single spider 8192kbps for whole Fetch		This is the byteflow ratefor the spider. Eg. To set the spiders to use a maximum of 64kbps each and 2Mbit in total: MaxKBPS=64 MaxGlobalKBPS=2048 NB: It is better to set this total somewhat lower in order not to kill your bandwidth.
IndexPath	./		Path to index files with respect to DRE
ImportPath	./		Path for import
DreHost	NA	Any IP	The engine (DRE) machine's IP address

[Default Spider] Section

Table 59 – [Default Spider Section] Section of DRE Configuration File

Key	Default	Range	Description
IndexPort	NA	Any port	The engine (DRE) indexing command port
Database	NA	NA	The database to use in the engine
ProxyHost	""		Host name or IP address of proxy server host to use
ProxyPort	80	Any port	Port number for proxy server
ProxyUsername	""	Any	Username for proxy server
ProxyPassword	""	Any	Password for proxy server
Directory	Spider name	Any	Directory into which to spider, the default is to use a subdirectory with the same name as the spider in its working directory

Key	Default	Range	Description
LogFile	None	Any	File for the spider to output logging into. This should only be used for debugging / tuning spiders, as log files become large very quickly
LogFileMaxSize	4MB	Any	The maximum size of the logfile in kilobytes.
Depth	99	0-512	Maximum depth to spider from the URL
TotalSize	1572864000 / 1.5 Gbytes /		This specifies the total size of the spider in Bytes. To set the fetch to get more than 2Gbyte for a single spider, set the TotalSize=10000mbytes and it will fetch 10 GBytes, 10000000kbytes will do the same.
SiteDuration	43200 / 12hours /	0-	aximum time to spend spidering the URL (sec.)
MaxPages	1E+08	1-	Maximum pages to retrieve from the URL
FollowRedirect	TRUE	TRUE FALSE	Accept and follow redirects. Often the spider will be redirected to another site (see below). NB: A redirect counts as another depth.
StayOnSite	TRUE	TRUE FALSE	Specifies whether the spider should only spider pages on the site with the same domain as the initial URL
PageDelay	0	0-	Specifies the 'politeness' of the spider. This will automatically ensure that page retrievals from a site will have an average delay of no less than this number of seconds between them.
MaxKBPS	4096		This specifies the maximum rate at which to transfer data for each spider.
MaxGlobalKBPS	8192		This specifies the maximum rate at which to transfer data for all spiders.
MinPageSize	200		Minimum page size in bytes (pages smaller than this will not be saved for indexing)
MaxPageSize	10000000		Maximum page size In bytes
MaxLinksPerPage	100		Maximum links per page (used to prevent indexing of index pages, they are however spidered as normal)
PageTimeout	45		Maximum time to spend on one page (including connect and request send). Failed pages will be retried 3 times with the same timeout (sec.)
PageDuration	3600 / 1 hour /		The time to spend on each page.
Extensions	SPIDER_ALLDOCS	Any	Comma separated list of wild cards specifying acceptable extensions to follow.

Key	Default	Range	Description
MustHaveCSVs	“”	Any	Comma separated list of wild card expressions that must appear for the page / link to be acceptable.
MustHaveCheck	0	1: URL 4: Header 8: Body 64: Case insensitive 128: Prevent Request	Where to search for the above list of expressions, OR these values together for multiple.
CantHaveCSVs	“”	Any	As above, but the page cannot have these if it is to be accepted
CantHaveCheck	SPIDER_PAGEHEADER	1: URL 4: Header 8: Body	As above.
DateCheck	0	0: Date last modified 1: URL 4: Header 8: Body 64:Case insensitive 128:Prevent Request	As above, specifying where to check for a date
AfterDate	-9000	0 - -9000	Page must have been modified within this many days from today (negative is backward in time)
BeforeDate	0	0 - 3000	File must be older than this number of days.

Key	Default	Range	Description
DateFormats	“”	MM/DD/YYYY, SHORTMONTH-DD-YY DD-MM/YY, DD-LONGMONTH YYYY, etc.	Comma separated date formats. These format specifiers are used whenever dates can be specified in the DDMMYY format style strings. Format specifiers: YY – Year (2 digit) , e.g. 99 or 00 or 01 etc. YYYY - Year (4 digit), e.g. 1999 or 2000, 2001 etc LONGMONTH – January, March, August etc. SHORTMONTH -Jan, Mar, Aug etc. MM – Month (2 digit) 01, 10, 12 etc M+ - Month (1/2 digit) 1,2,3,10 etc. DD – Day (2 digit) 01, 02, 03, 12, 23 etc. D+ - Day (1/2 digit) 1, 2, 12, 13, 31 etc HH – Hour (2 digit) 01, 12, 13 etc. H= - Hour (1/2 digit) NN – Minute (2 digit) N+ - Minute (1/2 digit) SS – Second (2-digit) S+ - Second (1/2 digit) ZZZ – Time Zone (GMT, EST, PST, etc.) Examples: / / Dates with 1/2 digit days DateFormats=D+/SHORTMONTH YYYY, DDMMYY / / Quoted string to allow spaces and commas etc within the format DateFormats=”D+SHORTMONTH YYYY”, “Date: D+ LONGMONTH, YYYY” / / Directory style dates DateFormats+D+/M+/YY, MM/DD/YYYY etc. For more detail, please refer to the appendix
CheckBeforeDownload	FALSE	True / false	Checks the parameters which have been set. If the site is ruled out by these then the file will not be downloaded, and therefore no links from that page will be followed.

Key	Default	Range	Description
LoginMethod	SPIIDER_LGNNONE	FORM AUTHENTICATE	Login front page method type, FORM or AUTHENTICATE
LoginURL	NULL	Any	URL to use for FORM login page
LoginUserField	username	Any	Name of the login username field
LoginUserValue	“”	Any	Value to assign to username field
LoginPassField	password	Any	Name of the password field
LoginPassValue	“”	Any	Value to assign to the password field
LoginFieldName0		Any	Name of additional field
LoginFieldValue0		Any	Value of additional field
LoginFieldNameN		Any	Optionally supply ‘N’ fields
SecurityType		NONE SSL_V23 SSL_V3 SSL_V2 TLS_V1	Specifies the type of SSL used to retrieve https:/ / prefixed URLs. If you do not wish to spider secure areas of sites wither remove this or set to NONE.
CookieNameN		Any	Cookie name to use for cookie based logins
CookieValueN		Any	Cookie value to use
SpiderStartDeleteOld	FALSE	TRUE FALSE	Specifies whether to strip directories of downloaded files before another spider cycle takes place.
BatchProcess	IMPORT	IMPORT INDEX NONE	Specifies the batch process to perform (see section Error! Reference source not found.)
BatchSize	50	May-00	Specifies the batch size (number of documents downloaded before executing the batch process). If 0, batch processing is turned off.
FixedFieldName0	NA	Any String	Specify the name of a fixed value field to be added to retrieved documents indexed into the DRE
FixedFieldValue0	NA	Any String	Specify the value of a fixed value field to be added to retrieved documents indexed into the DRE
FieldName0	NA	Any String	Specify the name of a field to be added to retrieved documents indexed into the DRE

Key	Default	Range	Description
FieldStart0	NA	Any String	Specifies the string identifying the start of a field value to be added to retrieved documents indexed into the DRE
FieldStop0	NA	Any String	Specifies the string identifying the end of a field to be added to retrieved documents indexed into the DRE
FollowRobotProtocol	Jan-00		When this is set to1, it enables a process which searches for a text file declaring what sites can and cannot be fetched.
SpiderAs	Any		Default name with which a particular spider identifies itself to sites. (this controls what pages it may access under the robots.txt protocol)
URL	“”	Any string	This specifies the URL to start from.
URLFile	“”	Any string	This is the name of the file, which contains a list of URLs.
URLFileDelete		Any string	This deletes the URL file.
URLCaseSensitive		TRUE FALSE	This specifies whether or not the URL is case sensitive
HTTPVersion	HTTP_V10		Allows the fetch to specify to the Web Servers the http version it supports. E.g. HTTPVersion=HTTP / 1.0
DateLongMonthCSVs	(English long versions of Months)	January-December in corresponding language.	Specifies the LongMonth values. Case insensitive.
DateMonthCSVs	(English Short versions of month)	Jan-Dec in corresponding language.	Specifies the ShortMonth values. Case insensitive.
DatePostfixCSVs	(English versions of day postfixes. E.g. 1st,2nd etc.	st, nd, rd, th.	Specifies the day postfixes. Case insensitive.
StoreSiteStructure	0	Jan-00	Specifies whether the spider should store the structure of the Website in order to download pages that contain new information only.

Key	Default	Range	Description
SiteStructureFlags	MONTH: 128 (default)	SiteStructureFlags = ➢ HOUR: 16 ➢ > DAY: 32 ➢ > WEEK: 64 ➢ > MONTH: 128 (default) ➢ >PREVENT URLCHECK: 16384	Sets the maximum time before a URL is downloaded regardless of whether it is expected to have changed.
NavLinkDefStartCSVs			This defines the start of the html that defines the navigation links to follow as start / end pairs
NavLinkDefEndCSVs			This defines the end of the html that defines the navigation links to follow as start / end pairs
NavLinksToFollow	SPIDER_FOLLOW ALL	0 – Don't follow any links 1 – Frames 2 - Hrefs 4 – Location (Javascript location.href=) 8 – HttpEquiv (HTML redirects) SPIDER_FOLLO WALL	Indicates the type of links to follow
NavSiteCheck	129	1 – Check URL 64 – Case Insensitive 128 – Check before Download	Criteria for determining whether or not to spider the site
NavSiteAllowCSVs	""	Character String. Can include wildcards	Comma separated string of characters which must appear in the URL of the sites that are to be spidered Use \? and * to mean actual characters '?' and '*' as opposed to the wildcard entries ? and *

Key	Default	Range	Description
NavSiteDisallowCSVs	“”	Character String. Can include wildcards	Comma separated string of characters which must not appear in the URL of the sites that are to be spidered Use \? and * to mean actual characters '?' and '*' as opposed to the wildcard entries ? and *
NavDirCheck	129	1 – Check URL 64 – Case Insensitive 128 – Check before Download	Criteria for determining whether or not to spider the directories of the root site(s).
NavDirAllowCSVs	“”	Character String. Can include wildcards	Comma separated string of characters which must appear in the URL of the Site directories that are to be spidered. Use \? and * to mean actual characters '?' and '*' as opposed to the wildcard entries ? and *
NavDirDisallowCSVs	“”	Character String. Can include wildcards	Comma separated string of characters which must not appear in the URL of the site directories that are to be spidered. Use \? and * to mean actual characters '?' and '*' as opposed to the wildcard entries ? and *
LogFileMaxSize	4MB	numerical	Specifies the maximum size of .log file. Once it has reached this limit it is moved to a file called .log.previous and a new .log file is started. .log.previous is overwritten the next time this happens.
IndexOverSocket	FALSE	TRUE / ON / 1	Specifies that indexing is to be done over the socket.
ImportPath			Path to index files with respect to HTTPFetch

Appendix B: Detail of autoindexer.cfg Sections

[Configuration]

This section governs the basic Automatic Indexing operation. It has the following *key=value* pairs:

Table 60 – [Configuration] Section of the autoindexer.cfg File

Key	Description
PollingPeriod	This specifies the time interval in milliseconds between each Autoindexer poll. E.g. PollingPeriod=5000
PollingMethod	This specifies whether the files to be processed by the Autoindexer are read from a file or from a directory. The value of this parameter can be either: 1 for 'file' 2 for 'directory' E.g. PollingMethod=2 NB. When using Autoindexer to process a large number of files on NT, you must use File Polling and not Directory Polling. File Polling simply looks at the filenames listed in a file and then processes them (which is more scalable because it does not require any directory tree polling). On UNIX making a list of file names is easy. Use Is and send the output to a file. On NT this does not work so to make a directory listing on NT set the following: [configuration] FilenameOutputMode=1 in the configuration file and this prints out the list of all the filenames that Scan would process in directory mode. All settings for directory polling, recursion, musthaves, canthaves, dates etc. are taken into account when listing the filenames. Note, that this setting should go in the configuration file and not in each job section. This will only list the filenames in the directories of the first job (or the default one). After that it will quit. If you have more than one job, and want to create more than one listing of filenames, just move the appropriate job to the top and run it. This mode prints the filenames to stdout, so to save it into a file just pipe it out ("autoindexer.exe> fileslist.txt"
RemoveLogFileOnStart	This indicates whether or not to remove the Log file before running Autoindexer. The value for this parameter can be either: 1 or on for 'Yes' 0 or off for 'No' E.g. RemoveLogFileOnStart=0 The value of this Key=value pair defaults to false (i.e. 0 or no).
Number	Represents the number of jobs that are to be performed by a particular Autoindexer.

Key	Description
N	Represents the name of job number n. n=0 for the first job.
MaxLogKBytes	Represents the maximum size the log file can reach before it is renamed as .log.previous and a new one is produced. It is measured in Kilobytes.

[Default]

This section governs the default configuration settings. The values in this section are employed for any job stated in the [Configuration] section, that does not have its own definition entry.

Table 61 – [Default] Section of the autoindexer.cfg File

Key	Description
deleteEscapeReferences	Boolean value that when set to 1 will escape references when doing deletion. This is needed when trying to remove documents with spaces in the reference. You will need to use it with DRE version 3.0 and setting UNESCURL=1 in the DRE.INI
deletePathReplaceUpToSlash	If PollingAction=7 or PollingAction=8, this is used to specify a string that is to be replaced up to a certain'/'. This is used so that a single portion of many strings which contain different substrings can be replaced as opposed to just one particular word. E.g. In the case where the files in the queue are C:\a\b\hello,c:\a\c\hello And c:\b\a\hello deletePathReplaceUpToSlash=3 Would replace the portion up to the third back-slash in each string.
DeleteReferenceFromContent	Boolean value telling Autoindexer when it deletes files that the reference may have been obtained from the document content. The setting is either on (1) or off (0). Instead of using the usual file reference for example c:\data\myfile this setting enables the reference to be obtained from the document content.
DeleteReferenceStart	Start tag to be used when obtaining the reference from content. E.g. deleteReferenceStart=<DOCID>
DeleteReferenceEnd	End tag to be used when obtaining the reference from content. E.g. deleteReferenceStart=</DOCID> So using the deletereferencestart and end, the content found between the two will be used as the document reference. These settings only apply when deleting documents, as to delete all the Autoindexer does is send a delete command to the DRE with a reference. If you leave either of these start or end parameters blank then it defaults to beginning and end of file. You can also use \n, \r, \t in the settings so that you can delimit the reference by return characters.
directoryAfterDate	The number of days before today that the document must have been modified. E.g. directoryAfterDate=-1 (i.e. yesterday)

Key	Description
directoryBeforeDate	The number of days after today that the document must have been modified. E.g. directoryBeforeDate=5 (i.e. 5 days from today)
DirectoryCantHaveCSVs	This specifies the string that must not appear in the directory path of a spidered document. E.g. directoryCantHaveCSVs=*.sys,*.bat,*.exe
DirectoryFileMatch	This is a wild card specification of which files in the directory to process. E.g. directoryFileMatch=*
DirectoryMustHaveCSVs	This specifies the string that must appear in the directory path of a spidered document. E.g. directoryMustHaveCSVs=*/temp/*
DirectoryPathCSVs	This specifies the directory in which lie the files to be processed. E.g. directoryPathCSVs=D:\projects\autoindexer\files This can be a comma separated list of directories which means that more than one directory can be processed.
DirectoryPathRecurseMatchCSVs	This setting indicates a set of wildcards to match against whilst recursing the directory tree. This is different from directoryMustHaveCSVs in that the wildcard match is done against the recursion path not the full path of the file. Hence, it's more efficient as it won't recurse directories that it doesn't need to do. So for example: directoryPathCSVs=C:\files\ directoryPathRecurseMatchCSVs=*01*,*02* will process: c:\files\199901 c:\files\199902 It will not do: c:\files\morefiles\199901 c:\files\morefiles\199902 as the match is done at every recursive step.
DirectoryRecurse	This indicates whether or not to recurse into directories. It can be either: 1 or on for “Recurse” 0 or off for “Don’t Recurse” E.g. directoryRecurse=off
DREHost	IP address where the DRE resides. Should you require to index into more that one DRE separate the IP addresses by a comma. E.g. DreHost=localhost,120.7.0.0
FilenameOutputMode	Boolean value where, if set to 1, will create a file listing all the files that are to be processed. It is used so that if you need to do file based processing you can create a list of files that need processing from a directory.
FilePollFilename	This specifies the filename from which the list of files to be processed is read. E.g. filePollFilename=queu e
FileBaseDirectory	This is the directory path to attach to each file in filePollFilename. You can specify either the full paths or just the File names inside the queue file. E.g. BaseDirectory=c:\Files\

Key	Description
importIDXFilesAction	Specifies what to do with idx files that have been processed. This can be either: 0: delete IDX file 1: move the IDX file to another directory 2: leave in directory
importIDXFielsMoveTo	If importIDXFilesAction=1, this is the full path that specifies where the idx files are moved to.
ImportPathReplaceString	Specifies the string that will replace the substrings of files in the queue.
IndexOverSocket	This indicates if the file to be indexed is to be sent over the socket or if it is local, see IndexLocalFile
IndexPort	The port number by which indexing commands are sent to the DRE. If this line is not present or is set to 0, then indexing will not be available. If you are using more than one DRE specify the index ports separated by commas. E.g. INDEXPORT=2001,6001
MoveToDirectory	This specifies which directory to move to when PollingPostAction is set to 2. E.g. MoveToDirectory=D:\projects\autoindexer\processed\
PollingAction	This specifies the action to perform when polling. The value of this parameter can be either: 1 for 'indexing idx files to a DRE' (idx files are files that are in hash form. See Section 4.1) 2 for 'importing files in various formats and indexing them into a DRE' 7 for import+index+delete this will import files to an IDX format, index them and also delete any documents from the DRE that have been removed from the local file structure 8 'delete from a DRE' E.g. PollingAction=2
PollingPostAction	This specifies the action to perform after the file has been processed. The value of this parameter can be either: 0 for 'do nothing' 1 for 'delete the file after processing' 2 for 'move the file to another directory'. It will keep the subdirectory structure. The value of this key=value pair defaults to 0. E.g. PollingPostAction=0 When moving the originally scanned files somewhere else, the subdirectory structure is retained. BOOL FileCopyKeepDirStructure(charsRoot1, charsFilename, charsRoot2, charsMode)
PollingMaxNumber	This specifies the maximum number of files to be processed at each poll. E.g. PollingMaxNumber=100 The value of this key=value pair defaults to 100.
QueryPort	The port number by which queries are sent to the DRE. Again if you are using more than one DRE specify the query ports separated by commas. E.g. QUERYPORT=2000,6000

Key	Description
IndexLocalFile	Specifies if the local IDX file is to be indexed or if it will be sent over the socket. You can set either IndexOverSocket=1 or IndexLocalFile=0 (this is the default) or you can set IndexOverSocket=0 or IndexLocalFile=1. The two pairs are equivalent. If both settings are present then IndexOverSocket takes precedence.
IndexLocalFilePathReplace IndexLocalFilePathString	When you are doing IndexOverSocket=off or IndexLocalFile=on (both are equivalent) - that is you are sending the filename to the DRE rather than the data over the socket, you need to tell it how to change the path so that if the IDX file is in c:\hello\a.idx, but the DRE is on another machine, so for the DRE it will be d:\hello\a.idx then you can set these two parameters
IndexMode=REFERENCE	Allows the killing of duplicates using methods other than the standard URL / reference match. Options include: REFERENCE – reference ONLY MATCHNN – conceptual match ONLY (i.e. if the documents are similar above a certain threshold NN) REFERENCE2MATCHNN – This means that it will check all databases for reference duplicates rather than the one database that is being indexed into.